

Matrix-free voxel-based finite element method for materials with heterogeneous microstructures

Matrixfreie voxelbasierte Finite-Elemente-Methode
für Materialien mit komplizierter Mikrostruktur

DISSERTATION

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

eingereicht an der
Fakultät Bauingenieurwesen
der Bauhaus-Universität Weimar

vorgelegt von
Dipl.-Ing. Andrea Keßler
geboren am 04. Mai 1980 in Erfurt
Weimar, Dezember 2017

1. Gutachter: Prof. Dr.-Ing. habil. Carsten Könke, Bauhaus-Universität Weimar
2. Gutachter: Prof. Dr. Andreas Meister, Universität Kassel
3. Gutachter: Prof. Dr. rer. nat. habil. Klaus Gürlebeck, Bauhaus-Universität Weimar

Tag der Disputation: 22.10.2018

Abstract

Modern image detection techniques such as micro computer tomography (μ CT), magnetic resonance imaging (MRI) and scanning electron microscopy (SEM) provide us with high resolution images of the microstructure of materials in a non-invasive and convenient way. They form the basis for the geometrical models of high-resolution analysis, so called image-based analysis. However especially in 3D, discretizations of these models reach easily the size of 100 Mill. degrees of freedoms and require extensive hardware resources in terms of main memory and computing power to solve the numerical model. Consequently, the focus of this work is to combine and adapt numerical solution methods to reduce the memory demand first and then the computation time and therewith enable an execution of the image-based analysis on modern computer desktops. Hence, the numerical model is a straightforward grid discretization of the voxel-based (pixels with a third dimension) geometry which omits the boundary detection algorithms and allows reduced storage of the finite element data structure and a matrix-free solution algorithm. This in turn reduce the effort of almost all applied grid-based solution techniques and results in memory efficient and numerically stable algorithms for the microstructural models. Two variants of the matrix-free algorithm are presented. The efficient iterative solution method of conjugate gradients is used with matrix-free applicable preconditioners such as the Jacobi and the especially suited multigrid method. The jagged material boundaries of the voxel-based mesh are smoothed through embedded boundary elements which contain different material information at the integration point and are integrated sub-cell wise though without additional boundary detection. The efficiency of the matrix-free methods can be retained.

Kurzfassung

Moderne bildgebende Verfahren wie Mikro-Computertomographie (μ CT), Magnetresonanztomographie (MRT) und Rasterelektronenmikroskopie (SEM) liefern nicht-invasiv hochauflösende Bilder der Mikrostruktur von Materialien. Sie bilden die Grundlage der geometrischen Modelle der hochauflösenden bildbasierten Analysis. Allerdings erreichen vor allem in 3D die Diskretisierungen dieser Modelle leicht die Größe von 100 Mill. Freiheitsgraden und erfordern umfangreiche Hardware-Ressourcen in Bezug auf Hauptspeicher und Rechenleistung, um das numerische Modell zu lösen. Der Fokus dieser Arbeit liegt daher darin, numerische Lösungsmethoden zu kombinieren und anzupassen, um den Speicherplatzbedarf und die Rechenzeit zu reduzieren und damit eine Ausführung der bildbasierten Analyse auf modernen Computer-Desktops zu ermöglichen. Daher ist als numerisches Modell eine einfache Gitterdiskretisierung der voxelbasierten (Pixel mit der Tiefe als dritten Dimension) Geometrie gewählt, die die Oberflächenerstellung weglässt und eine reduzierte Speicherung der finiten Elementen und einen matrixfreien Lösungsalgorithmus ermöglicht. Dies wiederum verringert den Aufwand von fast allen angewandten gitterbasierten Lösungsverfahren und führt zu Speichereffizienz und numerisch stabilen Algorithmen für die Mikrostrukturmodelle. Es werden zwei Varianten der Anpassung der matrixfreien Lösung präsentiert, die Element-für-Element Methode und eine Knoten-Kanten-Variante. Die Methode der konjugierten Gradienten in Kombination mit dem Mehrgitterverfahren als sehr effizienten Vorkonditionierer wird für den matrixfreien Lösungsalgorithmus adaptiert. Der stufige Verlauf der Materialgrenzen durch die voxelbasierte Diskretisierung wird durch Elemente geglättet, die am Integrationspunkt unterschiedliche Materialinformationen enthalten und über Teilzellen integriert werden (embedded boundary elements). Die Effizienz der matrixfreien Verfahren bleibt erhalten.

Contents

Contents	ix
Nomenclature	xi
1 Introduction	1
1.1 Project idea	1
1.2 Motivation	2
1.3 Intention of this work	5
2 Voxel-based model	7
2.1 State of the art	7
2.1.1 Geometrical model	9
2.2 Numerical model	10
2.3 Space filling curves	11
2.3.1 State of the art	11
2.3.2 Row-major order	11
2.3.3 Morton order	12
3 Finite element method	15
3.1 State of the art	15
3.2 Continuum mechanics	15
3.3 Numerical integration	19
3.3.1 Voxel-based integration technique	23
4 Iterative methods	24
4.1 State of the art	24
4.2 Error estimation and convergence assessment	25
4.2.1 Error estimation	25
4.2.2 Convergence assessment	26
4.3 Stationary methods	28
4.3.1 Jacobi method	28
4.3.2 Jacobi relaxation method	29
4.4 Conjugate gradients method	29
4.4.1 Preconditioned conjugate gradient method	31
4.4.2 Benchmark: Weighted Jacobi preconditioner	32

5	Matrix-free methods	34
5.1	State of the art	34
5.2	Adaption of the matrix-vector product	35
5.3	Element-by-element method	36
5.4	Node-edge based method	37
5.5	Examples: Academic and bone models	38
5.5.1	Benchmark: Matrix-free versus ANSYS solution	40
5.5.2	Benchmark: Storage versus calculation of associate node information in the EBE method	42
5.5.3	Benchmark: EBE versus NE method	42
5.5.4	Comparison of the solution time of the three models	43
5.5.5	Conclusion	44
6	Multigrid methods	45
6.1	State of the art	45
6.2	Introduction to the multigrid methods	46
6.2.1	Smoothing	48
6.2.2	Restriction and prolongation	48
6.2.3	Solution	49
6.3	Multigrid preconditioned conjugate gradient method	50
6.4	Examples: 3D-plate with hole and random distribution	51
6.5	Benchmark	52
6.5.1	Memory demand	52
6.5.2	MG vs. MGCG	53
6.5.3	Number of multigrid cycles	54
6.5.4	Number of multigrid levels	56
7	Mesh adaption based on octree algorithm	59
7.1	State of the art	59
7.2	Adaptive mesh construction	61
7.2.1	Saving of displacement constraints and hanging nodes on edges or planes in one integer	63
7.2.2	Solution algorithm considering hanging nodes	64
7.3	Example: Symmetric 3D-plate with circular hole	65
7.4	Benchmark: Space filling curve	65
7.5	Benchmark: Octree coarsening	67
8	Embedded boundary elements	69
8.1	State of the art	69
8.2	Embedded boundary method	71
8.3	Embedded boundary elements for voxel-based models	72
8.4	Example: 3D plate with circular hole	74
8.5	Benchmark	75
9	Summary, conclusion and outlook	78
A	Encode and decode morton numbers	81

B Bitwise calculation	83
Bibliography	84

Nomenclature

Abbreviations

2D	two dimensional
3D	three dimensional
μ CT	high resolution computer tomography or micro computer tomography
Alg.	algorithm
ANSYS	software - ANSYS Academic Research, Release 12.1, ANSYS Inc., www.ansys.com
ASCII	American Standard Code for Information Interchange
CG	conjugate gradient method
Ch.	chapter
CPU	central processing unit
CSR	compressed sparse row
CT	computer tomography
CUDA	Nvidia's parallel computing platform and application programming interface for access to the GPU
dof	degree of freedom
dofs	degrees of freedom
EBE	element-by-element
FCM	finite cell method
FEM	finite element method
Fig.	figure
GPU	graphic processing unit
HN	hanging node
IFEM	immersed finite element method
ITK	Insight Segmentation and Registration Toolkit
J	Jacobi method
JCG	Jacobi preconditioned conjugate gradient method
MG	multigrid
MGCG	multigrid preconditioned conjugate gradient method
Mill.	millions
min	minutes
MP	multiphase elements
MRI	Magnetic resonance imaging
NE	node-edge based
NuTo	numerical toolkit
PCG	preconditioned conjugate gradient method
Ref.	reference
rel.	relative

sec	seconds
Sec.	section
SEM	Scanning electron microscopy
Tab.	table
VTK	Visualization ToolKit
XFEM	extended finite element method

Capital Latin letters

A	arbitrary matrix
B	strain-displacement-operator
C	linear elastic material tensor
D	diagonal matrix
E	coordinate system tensor
I	unit matrix
J	Jacobian matrix
K	global stiffness matrix
K^e	element stiffness matrix
M	iteration matrix
P	preconditioning matrix
E	Young's modulus
F, G	arbitrary functions
N	finite element shape functions
P	Prolongation
R	Restriction
S	Smoothing
U	strain energie
V	volume
V^e	volume of element e

Small Latin letters

d	search direction vector
e	algebraic error
e_i	eigenvector
f	nodal force vector
n	outer normal unit vector
p	internal force vector
r	residual vector
u	displacement vector
v	approximate displacement vector
x	position in domain Ω
h, z	help vector
\bar{t}	surface force vector
\bar{u}	boundary displacement vector
dim	model dimension: number of voxels in one direction
e	element integer variable
h	mesh width
i, j, k, l, m	integer variables and indices

l	level number
n	number of nodes
n_e	number of elements
n_{ip}	number of integration points
p_n	polynomial of degree n
r	radial direction in polar coordinate system
s	node integer variable
x_i	scalar variable associated with coordinate system direction i

Capital Greek letters

Γ	total domain boundary surface
Γ^Ω	interface between Ω and Φ
$\Gamma^{\Phi \setminus \Omega}$	embedded domain boundary surface
Γ_f	force boundary surface
Γ_u	displacement boundary surface
Ω	domain of a body in physical space
Φ	embedded/fictitious domain
\mathbb{R}^3	three-dimensional physical space

Small Greek letters

$\alpha, \alpha_1, \alpha_2$	scalar values
β, β_1	
δ	Kronecker delta
ϵ	strain vector (or tensor)
γ	lateral strain vector (or tensor)
σ	stress vector (or tensor)
τ	lateral stress vector (or tensor)
ξ	position in local coordinate system
ζ	position in local coordinate system
κ	condition number
λ	eigenvalue
ν	Poisson's ratio
ω	weighting factor
ρ	spectral radius
$\sigma(\mathbf{K})$	spectrum of matrix \mathbf{K}
θ	rotational direction in polar coordinate system
ξ_i	scalar variable associated with local coordinate system direction i
ζ_i	scalar variable associated with local coordinate system direction i

Mathematical notations

$(\cdot)^{-1}$	inverse of (\cdot)
$(\cdot)^\top$	transpose of (\cdot)
$\mathbf{0}$	null vector
\cap	neither
\cup	and

NOMENCLATURE

δ	partial derivation
\emptyset	empty set
$\frac{\partial(\cdot)}{\partial x}$	partial derivative of (\cdot) with respect to x
$\int(\cdot) dx$	integral of (\cdot) in function of x
$\ \cdot\ $	Euclidean or quadratic norm
\neq	not

List of Figures

1.1	Mises stress for load case reanimation	1
1.2	Modeling of the sternum fixation technique with ANSYS on a macro-scale thorax model	2
1.3	Micro-scale voxel-based geometry of a sternum part with density color range (40 μm)	3
2.1	Trabecular bone (femoral head) with 19,5 μm voxel size, digital image data from [PB03]	10
2.2	A simple row order in two (left) and three (right) dimensions for cubic data	11
2.3	Row-based patch for an arbitrary node j in 3D	12
2.4	A first iteration Morton code based space filling curve in two (left) and three (right) dimensions	13
2.5	A second iteration Morton code based space filling curve in two (left) and three (right) dimensions	13
2.6	Morton key in 2D by interleaving bits depth first for 3-bit large numbers	14
3.1	Regular or Cartesian grid with equivalent element stiffness with possible different Young's moduli E_i	23
4.1	Rel. solution time of the weighted JCG with different weighting factors vs. the CG	33
5.1	3D patch of the eight elements belonging to the node $n = 13$	39
5.2	Cube with cubical hole and bone of femoral neck biopsy [PB03] examples	40
5.3	Numbers of dofs of the uniform cube (<i>block</i>), the void cube in cube (<i>hole</i>) and the bone model (<i>bone</i>) versus model dimension	41
5.4	Comparison of storage or calculation of associate node information in EBE, results are relative to the greatest <i>hole</i> or <i>bone</i> model with associate node information calculation respectively	42
5.5	Comparison of solution time and memory demand relative to the respective EBE solution versus degrees of freedom (dofs)	43

5.6	Solution time relative to the EBE block model (dim=200) versus degrees of freedom (dofs)	43
6.1	Multigrid construction of three grid levels in 2D	46
6.2	Different types of multigrid cycles: V-cycle, variant of W-cycle and sawtooth-cycle respective	47
6.3	Multigrid V-cycle from coarse to fine grid and back	48
6.4	Restriction (left) and prolongation (right) in 2D from finer to coarser grid	49
6.6	Degrees of freedom for the random distributed material (<i>random</i>) and the 3D-plate with circular hole (<i>plate</i>) for different model discretizations, given in voxels per dimension (dim)	51
6.5	Three dimensional plate with circular hole with 128 voxels per dimension (left) and cube with 75% random material voxel-based elements with 32 voxels per dimension (right)	52
6.7	Computation time of MG and MGCG relative to JCG method for the <i>plate</i> example with 256 voxels per direction	54
6.8	Computation time vs. number of multigrid cycles (nbrCycles) relative to JCG analysis for <i>random</i> example with 4 discretization, n_S is the number of applied smoothing steps	55
6.9	Computation time vs. number of multigrid cycles (nbrCycles) relative to JCG analysis for <i>plate</i> example with 4 discretization, n_S is the number of applied smoothing steps	56
6.10	Computation time vs. number of multigrid cycles (nbrCycles) relative to JCG analysis for <i>plate</i> example with 4 discretization, number of multigrid levels is restricted to 3, n_S is the number of applied smoothing steps . . .	57
6.11	Computation time vs. grid levels relative to JCG analysis for <i>plate</i> example with discretization of 192 voxels per direction for grid levels from 2 (min) to 7 (max) grid levels for 1 to 2 V-cycle and 1 to 3 numbers of smoothing steps	58
6.12	Computation time vs. grid levels relative to JCG analysis for <i>plate</i> example with discretization of 192 voxels per direction for grid levels from 3 to 7 grid levels for 1 V-cycle and 1 to 5 numbers of smoothing steps	58
7.1	Resulting octree grid of a cube with void quarter tube with resolution $dim = 32$ and with colored element identifier	62
7.2	Quadtree: 2D grid with coarser middle element and associated hanging nodes (filled circles)	64
7.3	Degrees of freedom for the symmetric 3D-plate with circular hole (<i>sym</i>) for different model discretizations (dim)	65

7.4	Displacement solution for an octree construction with determined pre-requirements for $dim = 32$ model	66
7.5	Computation time and memory demand vs. dofs for the Morton sorting relative to row-major sorting solved with JCG, the example is the <i>plate</i> example with different discretizations	67
7.6	Dofs for the <i>sym</i> example with $dim=32$ and $dim=64$ with (oct-xL) and without (oct-1L) octree data structure	68
7.7	Rel. computation time and rel. memory demand for the row-major (row), Morton (o1L) and octree (oxL) based data structures, with the JCG analysis for the <i>sym</i> example with 32 and 64 voxels per direction relative to the larger row-based discretization	68
8.1	The domain Ω is embedded in Φ	71
8.2	The domain Ω embedded in the domain Φ (left), the corresponding pixel image (middle) and the coarse grid with two phases (right)	72
8.3	Example: Symmetric 3D-plate with circular hole	74
8.4	Stress reference solution for symmetric plate with hole ($a/w = 0.25$) for $\theta = 90^\circ$	75
8.5	Stress σ_y for the embedded boundary element model (left) and the respective single-phase model (right) with 32 voxels per dimension	76
8.6	Stress σ_{max} (left) and σ for $\theta = 90^\circ$ (right) at the hole for the embedded boundary element model and the original model for the <i>plate</i> example relative to the maximal stress $\sigma_{ref} = 3, 23$ of the analytical solution . . .	76
8.7	Computation time and memory demand vs. dofs of the embedded boundary element and the original model for the <i>plate</i> example	77
8.8	Computation time and memory demand vs. dimension of the embedded boundary element and the original model for the <i>plate</i> example	77

List of Tables

5.1	Results of ANSYS vs. NE and EBE solution in terms of computing time and main memory demand	41
6.1	Computation time and memory demand of the <i>plate</i> (dim=256) analysis with 41.1 Mill. dofs performed with the PCG solver of the commercial software ANSYS and with the proposed MGCG solver with 6 grid levels and one cycle and pre- and post-smoothing step	53
7.1	Displacement constraints integer: Bit-wise storing of constraint information and hanging node characteristic of one node	64

Chapter 1

Introduction

1.1 Project idea

The institute of structural mechanics (ISM) has worked together with the thorax surgery of the Friedrich-Schiller-Universität on modeling different closure techniques for median sternotomy, the stabilization of the sternum after a surgery at the thorax where the sternum is cut in two halves [Bru+06].

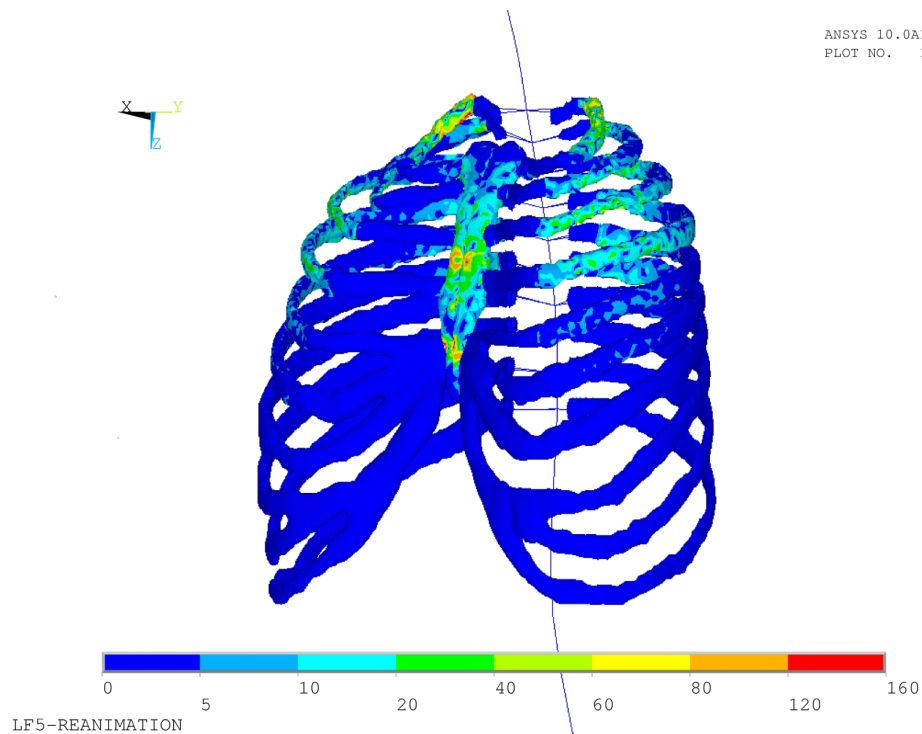


Figure 1.1: Mises stress for load case reanimation

Later on the cooperation was extended to the company Aesculap. For the development of the product SternumFIXation implant [Aes08] the ISM compared the new fixation technique with the common wire closure techniques of the sternum after median

sternotomy.

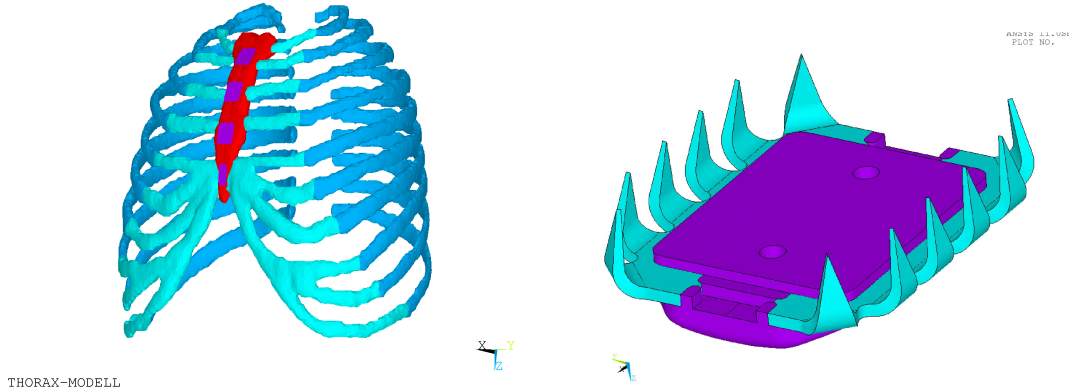


Figure 1.2: Modeling of the sternum fixation technique with ANSYS on a macro-scale thorax model

Closure techniques such as wires insert localized forces on mainly old bone which is optimized for completely different load cases. The local microstructure under the closure technique has an influence on the success of the application. The influence of the local material properties of the bone increases with the non-uniformity of the material properties distribution as in the case of older bones or osteoporosis. So the interest in modeling the underlying microstructure arises. The use of higher resolution image data seems straightforward considering modern image detection techniques such as micro computer tomography (μ CT), magnetic resonance imaging (MRI) and scanning electron microscopy (SEM) provide us with resolutions up to $0.4 \mu\text{m}$. Fig. 1.3 shows the microstructure of a part of a sternum with a resolution of $40 \mu\text{m}$. These considerations have been the starting point for this work.

1.2 Motivation

Traditionally complex material laws have been constructed to replicate the material response of materials with heterogeneous microstructure and the governing material variables of the mixture were filled by expensive experiments. Nevertheless this approach is restricted by the feasibility and accuracy of the experiments. With the increase of the computational powers and high resolution methods to detect geometrical material data at the micro-scale numerical simulations on smaller scales gained ground. The heterogeneous microstructure is modeled on smaller scales to obtain the bulk material behavior (hierarchical multi-scale models) or describes the material behavior in detail (micro-scale or concurrent multi-scale models). On a finer scale simpler material laws with intrinsic material parameters of the different materials can be applied. In this work the material will be modeled on the micro-scale based on high resolution image data.

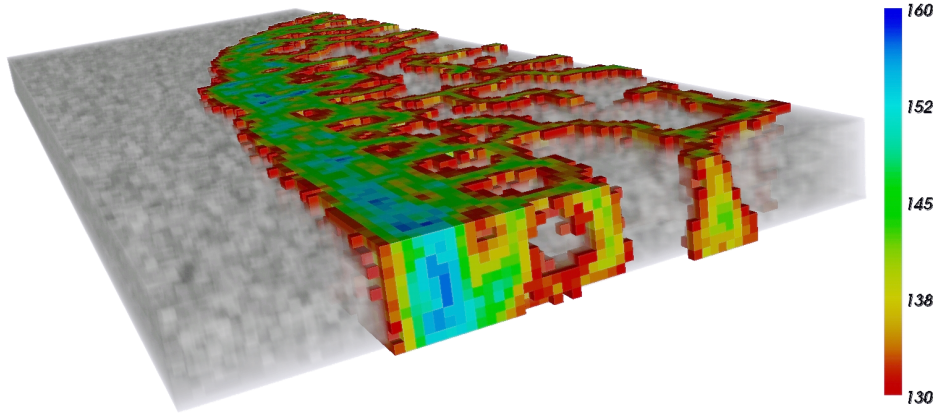


Figure 1.3: Micro-scale voxel-based geometry of a sternum part with density color range (40 μm)

Modern image detection techniques such as micro computer tomography (μCT), magnetic resonance imaging (MRI), ultrasound and scanning electron microscopy (SEM) provide us with high resolution images of the underlying microstructure. With image techniques we can detect the microstructure of the material non-invasive and convenient. The digital data is nowadays widely used in mechanics of materials, optics, medicine, biology and medicine techniques. In [Häf07; Sch13] different pictures of microstructures from digital image detection techniques are shown. The digital image color intensity is transferred to the material distribution and used to generate the geometry. E.g. for bone the material parameters are derived through the Hounsfield unit as a regression function. The ability of modern image detection techniques is also used for mainly homogeneous materials on micro-scale such as metal alloys in aero-construction. Here small micro-defects like voids can lead to macro-failure, so detection and modeling of the microstructure is of mayor importance.

Full micro-scale models based on high resolution digital image data with resolutions of 0.4-50 μm contain several million degrees of freedom (dofs) and reach the boundaries of today's computational power and storage capacities. Even for small parts (dimension of millimeters) a micro-scale model has about couple of 10 million degrees of freedom. So efficient solution methods stay in the scope of interest.

The generation of the image-based mesh can be done through two main classes with various subclasses. The first class applies boundary detection techniques for the mesh construction. The other creates the mesh directly from digital image. Different computer-aided design (CAD) algorithms are provided in commercial software to produce a mesh based on boundary detection. The well-known marching cube algorithm [LC87] generates iso-surfaces of the geometrical data. From the boundary detection either boundary conforming or non-conforming meshes can be generated.

The more traditional approach generates boundary-conform meshes with standard FEM. But the algorithms are time-consuming and the quality of such meshes depends on the initial boundary detection of the surface. Iterative quality checks of the mesh are required to prevent numerical unstable ill-shaped elements which are common. In more recently approaches the boundary detection is used for the construction of non-conforming meshes through level sets with e.g. the extended finite element method (XFEM) [Leg+10] and the finite cell method (FCM) [Düs+08]. The simpler meshing process is gained through more cumbersome and time-consuming numerical integration and calculation of the stiffness matrix.

The method chosen for this work produces a mesh directly from the digital image voxel-based geometry and is straightforward, automatic, fast and robust. The first voxel-based finite element model was developed by Keyak et al. [Key+90], one hexahedral element is used to model one voxel. Even without a smooth material boundary the voxel-based FE model predicts accurately the homogenized elastic material properties [Rie+95; Che+07] and the calculated Mises effective stress and strain distribution correspond well to experimental micro-damage regions [NCG05].

Eventually the regular grid structure of voxel-based discretization provides the possibility to apply efficient numerical techniques which reduce the high memory demand and the computational cost. The matrix-free solution approaches were first applied for image-based μ FE analysis in [HK94]. They reduce the memory footprint considerably and allow the computation of large models on modern desktop computers. The used efficient iterative solution method of conjugate gradients [Saa03] is the most suited for large scale image-based analysis. The convergence rate is highly dependent of a good preconditioner. Especially suited for grid-based and matrix-free analysis is the multigrid method as preconditioner for the conjugate gradient method (MGCG) [TO94]. Further reduction of memory demand can be achieved through coarsening of homogeneous regions of the digital image by octree algorithms [Sam89].

Contrary to the aligned meshing in standard FEM, the grid based methods describe the image data correctly but not the photographed object. The object is rendered with the image-based voxel data resulting in a saw-tooth boundary for voxel-based models. For the stress solution of the voxel-based analysis sharp edges pose numerical problems yielding in stress discontinuities with numerical induced stress peaks. To overcome this drawback the jagged boundary of the geometry can be smoothed. Though, the regular grid structure has to be retained, this can be done through non-conforming discretization methods. As the advantages of the matrix-free analysis should not be lost, the smoothing is realized through embedded boundary elements with different materials at the integration point and an special sub-cell-based integration as e.g. in the finite cell method [PDR07; Düs+08] for high order ansatz function.

1.3 Intention of this work

The intention of this work is to develop a numerical solution algorithm for image-based analysis which exploits the advantages of voxel-based discretization with the main focus on memory efficiency. The purpose is to achieve a robust, automatic and efficient (in terms of computation time and memory demand) image-based analysis based on 3D voxel-based micro finite element models.

Thus the following numerical techniques are implemented, adapted and combined in this thesis:

- For the analysis of large models the memory requirement is the relevant limit. For voxel-based meshes the regular grid structure with hexahedral elements of the same shape, size and orientation makes it possible to store only a basis element stiffness matrix for each Poisson's ratio for all elements which reduces the memory demand considerably. The construction and inversion of the global stiffness matrix which is a memory and time consuming process is replaced by matrix-free solution techniques, first used by Hughes et al. [HLW83].
- For large symmetric positive definite linear systems the efficient and well-known iterative method of conjugate gradients (CG) is the first choice [Saa03]. In combination with the matrix-free approach the construction of an efficient preconditioner is challenging as classic approaches require an assembled stiffness matrix, which is not available here. Multigrid levels can as well as the original grid mesh be solved by matrix-free procedures and so the multigrid method is implemented as an efficient preconditioner for the conjugate gradient method though the method also works as a standalone solver.
- Micro finite element (μ FE) models which model each voxel as hexahedral element are very large. To further reduce the memory demand the model size is reduced by coarsening of homogeneous regions through the reverse of a octree refinement algorithms. The octree algorithm is implemented based on a pointer-less representation which stores the octree nodes in a lookup table which is accessed through a Morton key.
- To overcome the drawback of the jagged geometry boundaries, embedded boundary elements are proposed. They retain the regular structure and diffuse the material boundary through modeling the different materials inside the element at the integration point level. The integration process is then adapted to the discontinuity of the variable to guaranty a correct Gaussian integration.

The remainder of this thesis is structured as follows: Chapter 2 presents the geometrical model and applied data sorting. Chapter 3 includes the theoretical notations

with respect to continuum mechanic and finite element method. The chapters 4, 5 and 6 refer to the iterative solvers, the matrix-free adaption and the multigrid method respectively. The coarsening of homogeneous regions through an octree algorithm is presented in Chapter 7. Embedded boundary elements are introduced in Chapter 8. The last chapter gives a summary, the concluding remarks and an outlook for potential future research activities.

Chapter 2

Voxel-based model

The microstructure of materials influences the effective material behavior through the physical properties of the individual material components as well as the different geometrical configurations. Digital image data gained from the real material is taken as input for the microstructural geometry model. The regular structure of the image data is transferred to the geometry model, generating a voxel-based model. The numerical model discretizes the voxel-based geometry retaining this structure, one element correspond then to one voxel. This chapter contains the transition from the digital image data to the input file for the numerical tools.

2.1 State of the art

The effective material response of complex microstructures or heterogeneous material is influenced through the physical properties of the individual phases as well as the geometrical configuration. Different strategies are followed to generate a geometrical model with detailed microscopic material description. An indirect approach is the unit-cell approach (also referred as representative volume elements) for materials with highly repetitive patterns (e.g. titan alloy [GKA07] or textile composites [KS03]). Artificial created models following the known manufacturing process for inclusion-matrix composites are developed e.g. for concrete [Häf+06; EK08]. A detailed review of different approaches to simulate material behavior taking into account the underlying micro- or meso-structure is presented in [MS01]. In this work the microscopic geometry model is directly generated from digital image data of the real material.

With the technical advances in image detection techniques such as micro computer tomography (μ CT), magnetic resonance imaging (MRI), ultrasonic and scanning electron microscopy (SEM) the possibilities of digital image-based applications have increased. High resolutions with up to $0.4\text{ }\mu\text{m}$ as well as large diameters with high contrast images are feasible. So digital image-based detection techniques are nowadays a well-known tool to gain information about heterogeneous 3D microstructures of materials. In various

disciplines image detection is employed, without a claim to be exhaustive: mechanics of materials, optics, medicine, biology and medicine techniques.

The generation of an image-based geometrical model for the finite element analysis can be done through two main approaches. The first approach applies boundary detection techniques for the model construction. The other creates the model directly from the digital image. Different computer-aided design (CAD) algorithms are provided in commercial software to produce a model based on boundary detection. The well-known marching cube algorithm [LC87] and marching tetrahedron algorithm [TPG98] generate iso-surfaces representing the boundaries.

From the geometrical model generated through boundary detection either boundary conforming or non-conforming meshes can be generated. The more traditional approach generates boundary-conform meshes with standard FEM, e.g. in [PZ09] for a model of vertebrae bone. But the algorithms are cumbersome, not entirely automatic and often are leading to ill-shaped elements unfit for numerical simulations [TW00]. The quality of such meshes depends on the initial boundary detection of the surface. Iterative quality checks of the mesh are required to prevent numerical unstable ill-shaped elements which are common. A hybrid approach which combines Cartesian grid for homogeneous parts with direct discretization for the complicated geometry solved by efficient adapted solution algorithms is addressed in [Sch13]. In other recent approaches the boundary detection is used for the construction of non-conforming meshes through level sets with e.g. the extended finite element method (XFEM) [Leg+10], the finite cell method (FCM) [Düs+08] and the composite finite element [Lie+09]. The simpler meshing process is gained through more cumbersome and time-consuming numerical integration and calculation of the stiffness matrix. In the XFEM extended degrees of freedom are added which cause a less banded structure of the stiffness matrix.

The methods which produce a mesh directly from the digital image data as voxel elements are more straightforward, automatic, fast and robust. The first voxel-based finite element model was introduced by Keyak et al. [Key+90], where one hexahedral element is used to model one voxel in standard FE. Voxel-based image data analysis is e.g. applied in [Mao+16] for individual (patient specific) models for osteoporosis prophylaxis and for μ FE of higher resolution images [FA11b].

One of the most used tools for processing digital image data is the Visualization ToolKit [SML] and corresponding Insight Segmentation and Registration ToolKit [Joh+]. The first provides the input data file and the visualization files of the output where as the second is used for the preparation of the image data through various filters. A survey of methods related to digital topology and geometry is presented in [SSB15]. The image processing and segmentation is not the focus of this work and it is assumed that the used digital image data is already prepared for the meshing process (e.g. through noise reduction, threshold and connectivity filters) providing an input data file with

clearly distinguished material information. The solution post-processing is rendered with Paraview [AGL05].

2.1.1 Geometrical model

A voxel is a hexahedron which corresponds to a 2D pixel with the third dimension as the length between two neighbor images of an image data stack. The pixel is a 2D image data point with a given size. Both, voxel and pixel contain the image data which consists of one value, which represents the color. With a one bit pixel/voxel only black or white can be represented, RGB colors allow around 16,7 Mill. different colors. The input image data consist of voxels and the vertices of the voxels are called grid points. The length of one voxel is determined by the image resolution.

Digital images need image processing and segmentation before they can be used for further modeling. Artifacts and white noise are disturbing the needed image data and segmentation processes detect the geometry. The image processing and segmentation is not the focus of this work and is assumed already done. The digital image data can be a stack of 2D images or every from VTK/ITK supported data file type. The image data is transformed to a VTK image data file called **StructuredPoints** [SML] as the input file for the numerical simulations in the software *NuTo*. The head of the file is defined as follows:

```
# vtk DataFile Version 3.0
Data file was generated by NuTo
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 6 6 6
SPACING 1 1 1
ORIGIN 0 0 0
CELL_DATA 216
SCALARS scalars int 1
LOOKUP_TABLE default
```

The specific values of the model have to be filled in the header. **DIMENSIONS** are the number of voxels in each direction. **SPACING** is the length of one voxel in each direction. **ORIGIN** is the starting point of the image and is not needed for the numerical implementation. **CELL_DATA** is the number of voxels. Additional there is also **POINT_DATA** as number of points or in computational terms the number of nodes which is needed for generating the solution output data. The body of the file consists only of the color value of the voxel which represent the respective material. For further information on the input data structure it is referred to [Joh+].

The solution data of the image-based analysis is transferred back to a VTK data file. These files can be visualized with any visualization tool which supports VTK, e.g. Paraview [AGL05].

2.2 Numerical model

The voxel or pixel image values give e.g. for bone the Hounsfield Unit (Fig. 1.3), which is correlated linear to the density and from there to the Young's modulus using empirical equations [Rue+12]. The direct meshing of the voxel-based geometry results in a Cartesian mesh with elements of the same shape, size and orientation as shown in Fig. 2.1.

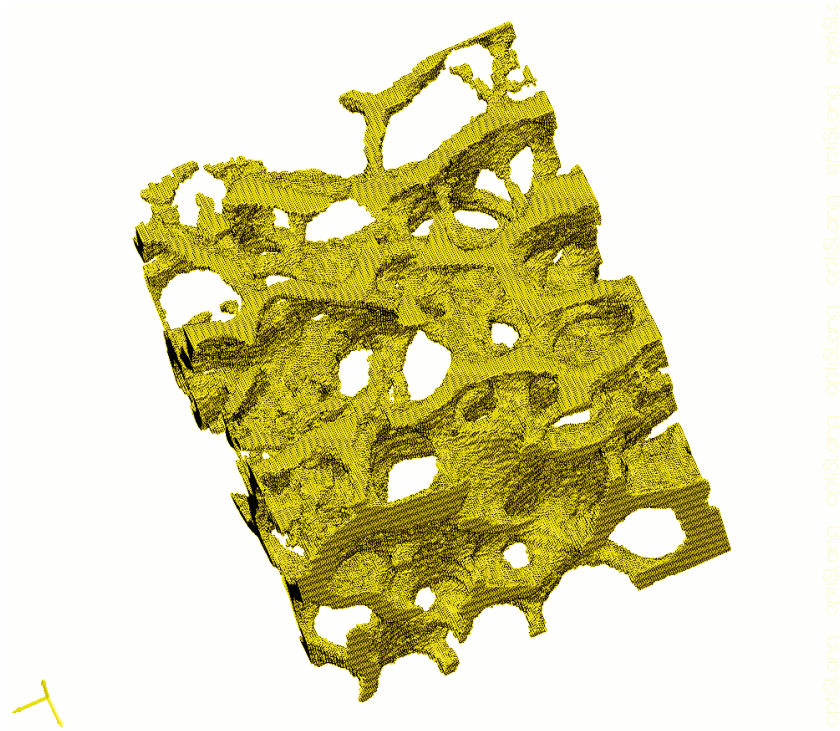


Figure 2.1: Trabecular bone (femoral head) with 19,5 μm voxel size, digital image data from [PB03]

Therewith the voxel-based numerical models allow a reduced storage of the finite element data structure, the element and node information is only stored in a few global vectors, eight 64-bit precision vectors are allocated: the displacements ($3n^3$), the residual ($3n^3$), the diagonal preconditioned vector ($3n^3$), Young's modulus ($(n-1)^3$), three 32-bit precision vectors for the identifying of the nodes and elements, one 1-bit vector for boundary conditions ($3n^3$) and the basis element stiffness matrix (24×24) and some additional values (here: n^3 is the number of nodes). This leads to only about 125 bytes per node. Only one basis element stiffness matrix for each Poisson's ratio has to be

stored for all elements which is multiplied by the individual Young's modulus due to the equally shaped and sized elements.

The sorting of the finite element data can be done with special space filling curves presented in the following section.

2.3 Space filling curves

2.3.1 State of the art

Data locality is an important condition for fast data access and highly efficient memory usage as the speed of the hierarchical memory access is one aspect which constrains the performance of fast computational solutions. Especially important is the data sorting through space filling curves for the multigrid and octree method as well as for efficient conversion to parallelization. E.g. in [GZ99] the parallelization is driven by space-filling curve domain decomposition for multigrid algorithm and in [MWZ06] a dynamically adaptive F-cycle is used for adaptively refined grids also in a multigrid algorithm.

Space filling curves basically map all points of a 2D or 3D region in a one dimensional curve. A historical review and introduction to different space filling curves is given in the textbooks [SH94; Bad12]. The first curve was found by [Pea90] in 1890. Many others followed, e.g. the well-known Hilbert curve [Hil91] and the Morton order [Mor66]. The later one is implemented in this work as it is well-suited for octree and multigrid construction [Lew+10; FA12]. Further the Hilbert curve and the Morton order can be used for generating partition algorithms as in [Lin+14] and [ABL16] respectively and the extension of the Morton code for tetrahedral meshes is developed in [BH16]. In [AW06] fast bitwise handling of integers for the Morton code is presented.

2.3.2 Row-major order

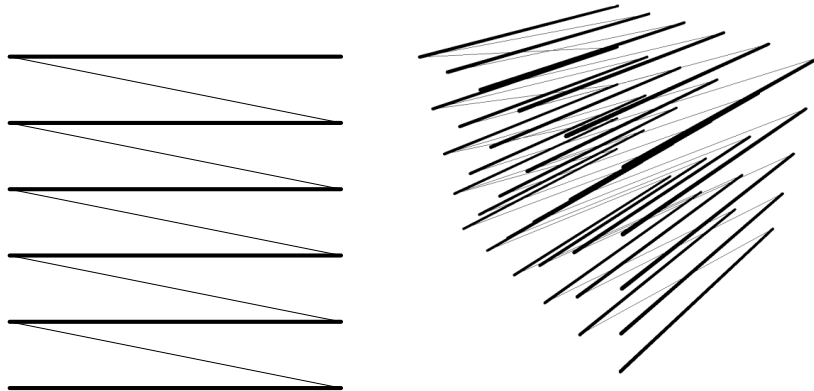


Figure 2.2: A simple row order in two (left) and three (right) dimensions for cubic data

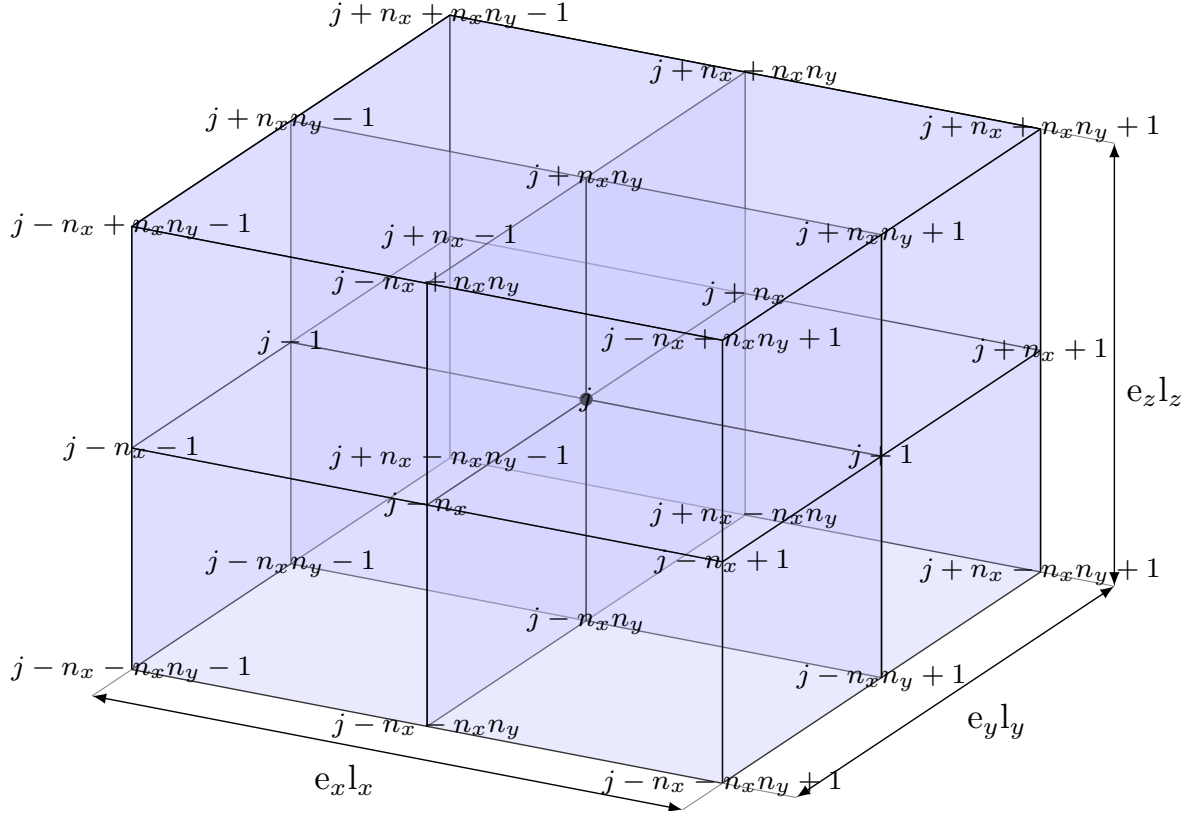


Figure 2.3: Row-based patch for an arbitrary node j in 3D

Especially for regular grids as well as image data the most common and straightforward method of data sorting is row- or column-major¹ based. In row-major the first row of the data is placed in contiguous memory, then follows the next and so on. The column neighbor is thus one row length away (Fig.2.2). For large models this distance is large and the data access is cache inefficient (cache-misses) and slower. For efficient (high-performance) computing the data transfer is a major bottleneck, so data locality is important. Nevertheless the computation of neighboring information through the element number such as nodes at the element or neighbor elements is simple and for all elements equal (Fig. 2.3) with additional ghost elements framing the cube model. For advanced sorting schemes these information have to be searched (binary search) which is computational more expensive.

2.3.3 Morton order

The Morton order is also called z-order because of the developing z-kind space-filling curve in which the data is sorted (Fig. 2.4 and 2.5). Each point in a quadrant is traversed before the curve goes to the next quadrant and this applies also to each quadrant itself. The so sorted data can then be accessed by search trees or lookup tables and is equivalent

¹Column-major is basically row-major with a coordinate system transformation.

to a depth-first traversal of a quadtree (2D) or octree (3D). Thus this sorting is especially suited for pointer-less octree or quadtree representation [Lew+10].

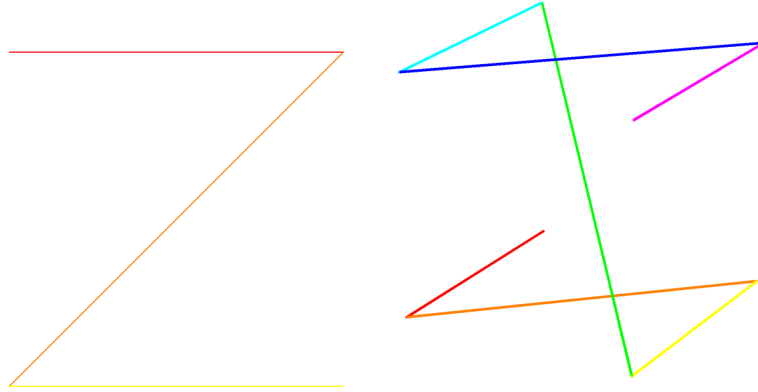


Figure 2.4: A first iteration Morton code based space filling curve in two (left) and three (right) dimensions

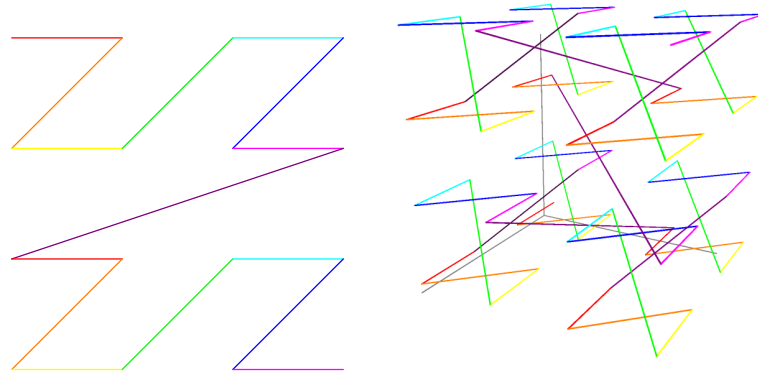


Figure 2.5: A second iteration Morton code based space filling curve in two (left) and three (right) dimensions

The sorting with the Morton key is done by interleaving the bits of the directions. The transfer from bits to integer numbers is done by multiplying the $(n)th$ bit by 2^n starting from $n = 0$:

```
8 4 2 1  -> a four bit number
1 1 0 1  -> 8*1+4*1+2*0+1*1=13
1 1 1 1  -> =15 - largest four bit number.
```

Interleaving bits means to interleave the number of each direction with the other directions depth first. The two 3-bit large numbers $x = 3$ and $y = 1$ results in a so called Morton key $m = 7$, where each bit is stored depth-first $y_3x_3y_2x_2y_1x_1$:

```

x: 3 :   0   1   1
y: 1 :  0   0   1
-----
m: 7 : 0 0 0 1 1 1 .

```

For 2D the bit interleaving to generate the Morton key is shown in Fig 2.6.

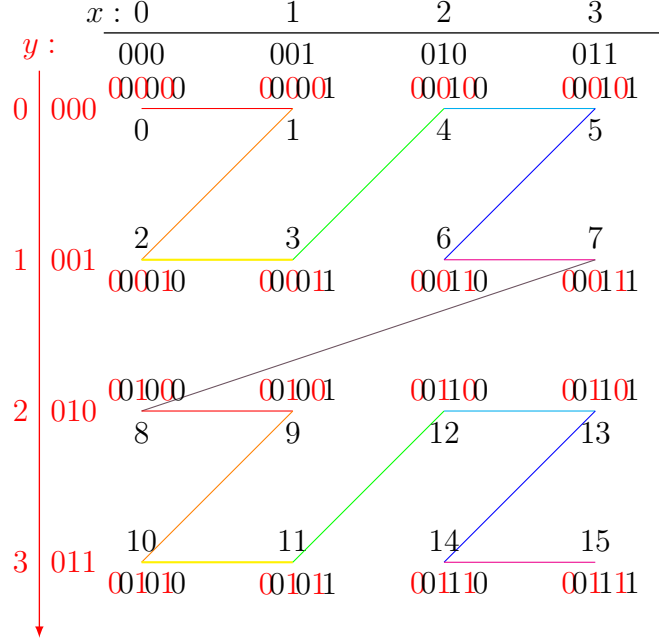


Figure 2.6: Morton key in 2D by interleaving bits depth first for 3-bit large numbers

The function for encoding and decoding Morton numbers can be found in the Appendix A.

For the regular finite element discretization the Morton key equally represents the elements and the nodes of the mesh, meaning each element has the same Morton key as the first node of this element. For the handling of integers with bitwise information e.g. for the Morton key calculation and in the octree algorithm bitwise calculation is necessary. The basic operations are shown in Appendix B.

With the presented Morton code the finite element data is stored efficiently, the local data is accessed locally in the solution algorithms due to the sorting, extensions to future partition and parallelization algorithms are straightforward and the basis for effective multigrid and octree algorithms is given.

Chapter 3

Finite element method

3.1 State of the art

Most applications in continuum mechanics can only be solved in the weak form of the differential equation system which is done by numerical discretization methods such as the finite element method (FEM). The concept of the continuum mechanic is described in the textbooks e.g. [Jog02; Spe04; BG13]. The FEM is one of the most general applicable numerical methods for models with arbitrary geometry and boundary conditions. More profound information can be found in the classical textbooks [Bat96; ZTZ05]. The FEM has been specified for many purposes, here it is adapted for voxel-based methods. [Sch13] uses the reusable basis element stiffness matrix to assemble the global stiffness matrix per nodal blocks. An efficient integration technique for voxel-based finite cell method is proposed in [Yan+12] which pre-computes large parts of the cell stiffness matrices. All matrix-free methods adapt the FEM for solving without the assembling of the global stiffness matrix (Ch. 5).

3.2 Continuum mechanics

The following notation policy is used in this work: Matrices (and tensors) are denoted by capital bold Latin letters (e.g. \mathbf{A}) or in special cases such as for the stress or strain notation in bold Greek letters ($\boldsymbol{\sigma}$ or $\boldsymbol{\epsilon}$ respectively). Vectors are denoted by small and bold Latin letters (e.g. \mathbf{v}) and scalars by small italic Greek letters (e.g. α) or small italic Latin letters for indices and counters mostly (e.g. i). For some scalars with mechanical meaning (e.g. E) and for the components of a matrix capital Latin letters are used. The scalar component of a matrix in the i -nth row and j -nth column is then e.g. A_{ij} . The i -nth component of a vector is referred by e.g. v_i .

A three-dimensional body is defined by a set of points \mathbf{x} which occupy a domain $\Omega \in \mathbb{R}^3$. The internal force \mathbf{p} is given inside the domain. Surface forces $\bar{\mathbf{t}}$ and boundary displace-

3. FINITE ELEMENT METHOD

ments $\bar{\mathbf{u}}$ are applied on the boundary surfaces Γ_f and Γ_u respectively with $\Gamma_f \cup \Gamma_u = \Gamma$ and $\Gamma_f \cap \Gamma_u = \emptyset$, where Γ is the total domain boundary surface. The unknown displacement field $\tilde{\mathbf{u}}$ defines the motion of the body under the applied loading condition. Considering small displacements the governing differential equations are the following:

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl} \quad \forall \mathbf{x} \in \Omega \dots \text{constitutive equations} \quad (3.1)$$

$$\epsilon_{ij} = \frac{1}{2} \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) \quad \forall \mathbf{x} \in \Omega \dots \text{kinematic equations} \quad (3.2)$$

$$\frac{\partial \sigma_{ij}}{\partial x_j} = b_i \quad \forall \mathbf{x} \in \Omega \dots \text{static equations} \quad (3.3)$$

$$\tilde{u}_i = \bar{u}_i \quad \forall \mathbf{x} \in \Gamma_u \dots \text{Dirichlet boundary condition} \quad (3.4)$$

$$\sigma_{ij}n_j = \bar{t}_i \quad \mathbf{n} \perp \Gamma \quad \forall \mathbf{x} \in \Gamma_f \dots \text{Neumann boundary condition.} \quad (3.5)$$

where \mathbf{n} is the outer normal unit vector. For an isotropic linear elastic material the generalized Hooke's law is applied and gives the material tensor

$$C_{ijkl} = \lambda \delta_{ij} \delta_{kl} + \mu (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}), \quad (3.6)$$

where δ denotes the Kronecker delta which is defined as

$$\delta_{ij} = \begin{cases} 1 & : i = j \\ 0 & : i \neq j \end{cases} \quad (3.7)$$

and λ and μ are the Lamé's constants. For linear material behavior the specific strain energy U has a homogenous quadratic form

$$U(\boldsymbol{\epsilon}) = \int_{\Omega} \sigma_{ij} d\epsilon_{ij} = C_{ijkl} \int_{\Omega} \epsilon_{kl} d\epsilon_{ij} \quad (3.8)$$

$$= \frac{1}{2} C_{ijkl} \epsilon_{ij} \epsilon_{kl} = \frac{1}{2} \sigma_{ij} \epsilon_{ij}. \quad (3.9)$$

The used tensor notation for the second order stress and strain tensors

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} \end{bmatrix} \quad (3.10)$$

can be written in a vector notation due to the symmetry of the respective tensor:

$$\boldsymbol{\sigma} = \mathbf{C} \boldsymbol{\epsilon} \quad (3.11)$$

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \tau_{12} \\ \tau_{13} \\ \tau_{23} \end{bmatrix} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{bmatrix} = \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{12} \\ 2\epsilon_{13} \\ 2\epsilon_{23} \end{bmatrix} = \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ \gamma_{12} \\ \gamma_{13} \\ \gamma_{23} \end{bmatrix} \quad (3.12)$$

In the first vector of $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ are entries which are not generally used but clarify that in the component notation $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ can be written with one index.

Due to the symmetry of the stress and stain tensor the material tensor is for

$$C_{ijkl} = C_{jikl} = C_{ijlk} \quad (3.13)$$

symmetric and the fourth order tensor \mathbf{C} can be reduced to 6×6 -matrix with C_{ij} for $i = 1..6$ and $j = 1..6$. The commutative differentiation of the specific strain energy U

$$C_{ijkl} = \frac{\partial \sigma_{ij}}{\partial \epsilon_{kl}} = \frac{\partial^2 U}{\partial \epsilon_{kl} \partial \epsilon_{ij}} = \frac{\partial^2 U}{\partial \epsilon_{ij} \partial \epsilon_{kl}} = C_{klij} \quad (3.14)$$

leads to a symmetric elasticity matrix $C_{ij} = C_{ji}$.

The Hooke's law is now

$$\begin{aligned} \sigma_{ij} &= C_{ijkl} \epsilon_{kl} \\ &= \lambda \delta_{ij} \delta_{kl} \epsilon_{kl} + \mu (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}) \epsilon_{kl} \\ &= \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij}. \end{aligned} \quad (3.15)$$

Respectively the stress and strain can be expressed as

$$\begin{aligned} \sigma_{ij} &= \left(\lambda + \frac{2}{3} \mu \right) \epsilon_{kk} \delta_{ij} + 2\mu E_{ij} \\ &= \frac{1}{3} \sigma_{kk} \delta_{ij} + 2\mu E_{ij} \end{aligned} \quad (3.16)$$

and

$$\epsilon_{ij} = \frac{1}{3} \epsilon_{kk} \delta_{ij} + E_{ij} \quad (3.17)$$

where \mathbf{E} is a coordinate system basis (tensor). These formulations are well suited to

3. FINITE ELEMENT METHOD

invert the equation system and express the strains in function of the stresses

$$\begin{aligned}\epsilon_{ij} &= \frac{1}{9\lambda + 6\mu} \sigma_{kk} \delta_{ij} + \frac{1}{2\mu} \left(\sigma_{ij} - \frac{1}{3} \sigma_{kk} \delta_{ij} \right) \\ &= -\frac{\lambda}{2\mu(3\lambda + 2\mu)} \sigma_{kk} \delta_{ij} + \frac{1}{2\mu} \sigma_{ij}.\end{aligned}\quad (3.18)$$

Exemplary for the stress in the first direction the correlation between the Lamé's constants and the Young's modulus E and the Poisson's ratio ν which are more common in engineering are shown:

$$\epsilon_{11} = -\frac{\lambda}{2\mu(3\lambda + 2\mu)} \sigma_{11} + \frac{1}{2\mu} \sigma_{11} = \frac{1}{E} \sigma_{11} \quad (3.19)$$

$$\epsilon_{12} = -\frac{\lambda}{2\mu(3\lambda + 2\mu)} \sigma_{11} = -\frac{\nu}{E} \sigma_{11}. \quad (3.20)$$

It yields

$$\lambda = \frac{E\nu}{(1 + \nu)(1 - 2\nu)} \quad \mu = \frac{E}{2(1 + \nu)} \quad (3.21)$$

and the symmetric material matrix

$$\mathbf{C} = \frac{E}{(1 + \nu)(1 - 2\nu)} \begin{bmatrix} 1 - \nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1 - \nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1 - \nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}. \quad (3.22)$$

The Hooke's law can then be written as

$$\epsilon_{ij} = -\frac{\nu}{E} \sigma_{kk} \delta_{ij} + \frac{1 + \nu}{E} \sigma_{ij}. \quad (3.23)$$

Reformulation the specific strain energy accordingly [BG13] using (3.16) and (3.17) and the bulk modulus

$$K = \lambda + \frac{2}{3}\mu = \frac{E}{3(1 - 2\nu)} \quad (3.24)$$

it is

$$\begin{aligned}
 U(\boldsymbol{\epsilon}) &= \frac{1}{2} \left(\frac{1}{3} \sigma_{kk} \delta_{ij} + \frac{1}{2\mu} E_{ij} \right) \left(\frac{1}{2} \epsilon_{kk} \delta_{ij} + E_{ij} \right) \\
 &= \frac{1}{6} \sigma_{kk} \epsilon_{ll} + \mu E_{ij} E_{ij} \\
 &= \frac{K}{2} \epsilon_{kk}^2 + \mu E_{ij} E_{ij}.
 \end{aligned} \tag{3.25}$$

The specific strain energy has to be positive definite: $U > 0$ for all strains $\epsilon_{ij} \neq 0$. Therefore both parts of the Eq. (3.25) have to be positive definite

$$\frac{K}{2} \epsilon_{kk}^2 > 0 \quad \forall \epsilon_{kk} \neq 0 \tag{3.26}$$

$$\mu E_{ij} E_{ij} > 0 \quad \forall \mathbf{E} \neq \mathbf{0} \tag{3.27}$$

This is true only if

$$K = \frac{E}{3(1-2\nu)} > 0 \quad \mu = \frac{E}{2(1+\nu)} > 0 \tag{3.28}$$

and hence $E > 0$ and $-1 \leq \nu \leq 0.5$ which is true for the most linear elastic materials and always true in the context of this work.

Variational principles considering the law of consistent conversion of energy lead to the weak forms of the differential equations. The principle of virtual displacements (virtual work) considering the differential equations (3.1) to (3.5) is given as

$$\begin{aligned}
 \int_{\Omega} \sigma_i \delta \epsilon_i \, d\Omega &= \int_{\Omega} p_i \delta \tilde{u}_i \, d\Omega + \int_{\Gamma_f} \bar{t}_i \delta \tilde{u}_i \, d\Gamma_f \\
 &: \tilde{u}_i = \bar{u}_i \text{ and } \delta \tilde{u}_i = 0 \quad \forall \mathbf{x} \in \Gamma_u.
 \end{aligned} \tag{3.29}$$

Both formulations are equivalent, if (3.29) is fulfilled for any kinematically compatible virtual displacement [Bat96]. Such solution exists only for a few geometrical simple problems. For most applications only the weak form of the differential equation system can be solved approximately by a numerical discretization method such as the finite element method (FEM).

3.3 Numerical integration

The domain Ω is subdivided in finite elements e which are defined by a particular number of nodes n . The displacement field \tilde{u} is approximated by a linear combination of element

shape functions N multiplied by the nodal displacement values \mathbf{u} at the nodes

$$\tilde{u}(\mathbf{x}) = \sum_{d=1}^n N^d(\mathbf{x}) u^d \quad \forall \mathbf{x} \in \Omega^e \quad (3.30)$$

with the following properties for the shape functions in standard displacement based finite elements where s is a specific node of a n -node element with the volume Ω^e :

$$\sum_{d=1}^n N^d(\mathbf{x}) = 1 \quad \forall \mathbf{x} \in \Omega^e \quad (3.31)$$

$$N^s(x_s) = 1 \quad N^{d \neq s}(x_d) = 0. \quad (3.32)$$

Introducing the strain-displacement-operator

$$\mathbf{B}^d = \begin{bmatrix} \partial N^d / \partial x_1 & 0 & 0 \\ 0 & \partial N^d / \partial x_2 & 0 \\ 0 & 0 & \partial N^d / \partial x_3 \\ \partial N^d / \partial x_2 & \partial N^d / \partial x_1 & 0 \\ 0 & \partial N^d / \partial x_3 & \partial N^d / \partial x_2 \\ \partial N^d / \partial x_3 & 0 & \partial N^d / \partial x_1 \end{bmatrix} \quad (3.33)$$

equation (3.2) reads

$$\epsilon_i(\mathbf{x}) = \frac{1}{2} \sum_{d=1}^n B_{ij}^d(\mathbf{x}) u_j^d. \quad (3.34)$$

By substitution of (3.34) in (3.11) the approximate stress field is obtained by

$$\sigma_i = \frac{1}{2} \sum_{d=1}^n C_{il}(\mathbf{x}) B_{lj}^d(\mathbf{x}) u_j^d \quad (3.35)$$

and introducing the discretized displacement, strain and stress functions in the principle of virtual displacement (3.29) yield the discretized weak form

$$\int_{\Omega} \delta u_i B_{ki}(\mathbf{x}) C_{kl} B_{lj}(\mathbf{x}) u_j \, d\Omega = \int_{\Omega} \delta u_i N_{ji} p_j(\mathbf{x}) \, d\Omega + \int_{\Gamma_f} \delta u_i N_{ji}(\mathbf{x}) \bar{t}_j(\mathbf{x}) \, d\Gamma_f. \quad (3.36)$$

The entries in the vectors \mathbf{u} and $\delta \mathbf{u}$ are constants and can be extracted from the integral and with arbitrary virtual displacement vector $\delta \mathbf{u}$ the equation (3.36) is only satisfied, if:

$$K_{ij} u_j = f_i \quad (3.37)$$

with the components of the element stiffness matrix

$$K_{ij} = \int_{\Omega} B_{ki}(\mathbf{x}) C_{kl} B_{lj}(\mathbf{x}) d\Omega \quad (3.38)$$

and the components of the equivalent element force vector

$$f_i = \int_{\Omega} N_{ji} p_j(\mathbf{x}) d\Omega + \int_{\Gamma_f} N_{ji}(\mathbf{x}) \bar{t}_j(\mathbf{x}) d\Gamma_f. \quad (3.39)$$

Sorting the unknown displacement values of all nodes in a global vector and assembling the element stiffness matrices and the element force vectors accordingly yield

$$\mathbf{K}\mathbf{u} = \mathbf{f} \quad \text{with} \quad (3.40)$$

$$\mathbf{K} = \int_{\Omega} \mathbf{B}^T \mathbf{C} \mathbf{B} d\Omega \quad (3.41)$$

$$\mathbf{f} = \int_{\Omega} \mathbf{N}^T \mathbf{p} d\Omega + \int_{\Gamma_f} \mathbf{N}^T \mathbf{t} d\Gamma_f. \quad (3.42)$$

The resulting element stiffness matrix is symmetric $K_{ij} = K_{ji} \quad \forall i, j = 1..n$ since the elastic material matrix \mathbf{C} is symmetric (3.14):

$$\begin{aligned} K_{ij} &= \int_{\Omega} B_{ki}(\mathbf{x}) C_{kl} B_{lj}(\mathbf{x}) d\Omega \\ K_{ij} &= \int_{\Omega} B_{jl}(\mathbf{x}) (B_{ki}(\mathbf{x}) C_{kl})^T d\Omega; \\ K_{ij} &= \int_{\Omega} B_{jl}(\mathbf{x}) C_{lk} B_{ik}(\mathbf{x}) d\Omega = K_{ji}. \end{aligned}$$

The resulting global stiffness matrix is also symmetric and has a sparse and banded character resulting from the local range of influence of the element shape function. Furthermore for a sufficient constraint equation system (Dirichlet boundary condition) the stiffness matrix is positive definite because the specific elastic energy is positive definite for $E > 0$ and $-1 \leq \nu \leq 0.5$. With (3.34) it is

$$U = \frac{1}{2} \boldsymbol{\sigma} \boldsymbol{\epsilon} = \frac{1}{2} \mathbf{u}^T \mathbf{B}^T \mathbf{C} \mathbf{B} \mathbf{u} = \frac{1}{2} \mathbf{u}^T \mathbf{K} \mathbf{u} > 0. \quad (3.43)$$

For isoparametric finite elements in 3D the coordinates and the displacement field are defined by the same interpolation scheme respecting the shape functions N of the used finite element type

$$\mathbf{x}(\boldsymbol{\xi}) = \sum_{d=1}^n N^d(\boldsymbol{\xi}) \mathbf{x}^d \quad \mathbf{u}(\boldsymbol{\xi}) = \sum_{d=1}^n N^d(\boldsymbol{\xi}) \mathbf{u}^d \quad (3.44)$$

with n as the number of integration points in the element. The derivatives of the dis-

placement field are given with

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \sum_{d=1}^n \frac{\partial N^d(\boldsymbol{\xi})}{\partial \mathbf{x}} \mathbf{u}^d. \quad (3.45)$$

The computation of the derivatives requires the transformation from the local coordinates $\boldsymbol{\xi}$ into a global space of coordinates where the displacement field is defined:

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{\xi}}{\partial \mathbf{x}} = \left(\sum_{d=1}^n \frac{\partial N^d(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \mathbf{u}^d \right) \frac{\partial \boldsymbol{\xi}}{\partial \mathbf{x}} \quad : \boldsymbol{\xi} = \{\xi_1, \xi_2, \xi_3\}. \quad (3.46)$$

The derivatives of $\boldsymbol{\xi}$ can be determined with regard to the interpolation function (3.44) by the so-called Jacobi transformation

$$\frac{\partial \boldsymbol{\xi}}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \right)^{-1} = \left(\sum_{d=1}^n \frac{\partial N^d(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \mathbf{x}^d \right)^{-1} = \mathbf{J}(\boldsymbol{\xi})^{-1}. \quad (3.47)$$

where \mathbf{J} is the Jacobi matrix

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_1}{\partial \xi_1} & \frac{\partial x_2}{\partial \xi_1} & \frac{\partial x_3}{\partial \xi_1} \\ \frac{\partial x_1}{\partial \xi_2} & \frac{\partial x_2}{\partial \xi_2} & \frac{\partial x_3}{\partial \xi_2} \\ \frac{\partial x_1}{\partial \xi_3} & \frac{\partial x_2}{\partial \xi_3} & \frac{\partial x_3}{\partial \xi_3} \end{bmatrix} \quad \mathbf{J}^{-1} = \begin{bmatrix} \frac{\partial \xi_1}{\partial x_1} & \frac{\partial \xi_2}{\partial x_1} & \frac{\partial \xi_3}{\partial x_1} \\ \frac{\partial \xi_1}{\partial x_2} & \frac{\partial \xi_2}{\partial x_2} & \frac{\partial \xi_3}{\partial x_2} \\ \frac{\partial \xi_1}{\partial x_3} & \frac{\partial \xi_2}{\partial x_3} & \frac{\partial \xi_3}{\partial x_3} \end{bmatrix} \quad (3.48)$$

Rewriting the element stiffness matrix in the local natural coordinate system gives

$$K_{ij} = \int_{\xi_1} \int_{\xi_2} \int_{\xi_3} B_{ki}(\boldsymbol{\xi}) C_{kl}(\boldsymbol{\xi}) B_{lj}(\boldsymbol{\xi}) \det \mathbf{J}(\boldsymbol{\xi}) d\xi_1 d\xi_2 d\xi_3 \quad (3.49)$$

and for the numerical approximation using n_{ip} integration points and the corresponding weighting factors ω replaces the integrals with the sum

$$\mathbf{K} = \sum_{i=1}^{n_{ip}} \omega^i \mathbf{B}^T(\boldsymbol{\xi}^i) \mathbf{C}(\boldsymbol{\xi}^i) \mathbf{B}(\boldsymbol{\xi}^i) \det \mathbf{J}(\boldsymbol{\xi}^i) \quad (3.50)$$

where $\boldsymbol{\xi}^i$ are the natural coordinates of the integration point i with the respective weighting factor ω^i . The most common weighting factors for solid elements are obtained by the Gaussian quadrature which integrates exactly a polynomial of the degree $2p - 1$ with a p Gauss points [Bat96]. Distorted elements can not be integrated exactly by a Gaussian quadrature and introduce an error which gets bigger the more distorted the element gets. Modeling the complex problem entirely with voxel elements this error is omitted.

The shape functions for the standard hexahedral elements with n nodes following the

isoparametric concept for the linear ($n = 8$) hexahedral element are defined by

$$N^i(\xi_1, \xi_2, \xi_3) = \frac{1}{2}(1 + \xi_1^i \xi_1) \frac{1}{2}(1 + \xi_2^i \xi_2) \frac{1}{2}(1 + \xi_3^i \xi_3) \quad : i = 1, \dots, n \quad (3.51)$$

and used as such for the implementation.

3.3.1 Voxel-based integration technique

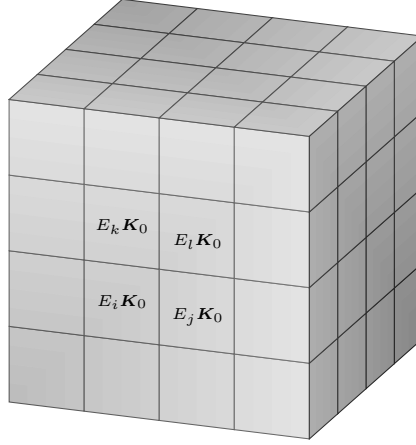


Figure 3.1: Regular or Cartesian grid with equivalent element stiffness with possible different Young's moduli E_i

If the finite element mesh consists of a Cartesian grid, which is the case in voxel-based discretizations (Fig. 3.1), the resulting elements are unit cubes with possible different element length in each Cartesian direction, but constant respective lengths over all elements. For such elements the Jacobi transformation matrix is constant and for all elements identical and hence the element stiffness matrix is identical for equal materials. For isotropic material and assuming a constant Poisson's ratio for all elements the element stiffness matrix has to be calculated only once even for varying Young's modulus. Eq. (3.41) can then be rewritten as

$$\mathbf{K} = \int_{\Omega} \mathbf{B}^T \mathbf{C} \mathbf{B} d\Omega = E \int_{\Omega} \mathbf{B}^T \mathbf{C}_0 \mathbf{B} d\Omega = E \mathbf{K}_0 \quad (3.52)$$

where \mathbf{K}_0 is the unified basic stiffness matrix computed with a Young's Modulus $E_0 = 1$ which is multiplied with E as distinct material property for each element. So in this work the basic element stiffness matrix is stored and re-used in the matrix-free solution process (Ch. 5). The computational effort of the integration of the element stiffness matrices is reduced by $\frac{1}{n_e}$ with n_e as the number of elements in the model and it is zero for assembling the global stiffness matrix.

Chapter 4

Iterative methods

4.1 State of the art

There are different direct solution methods which are very efficient for the solution of smaller linear equation systems, nevertheless their importance subsides for large systems, where iterative solvers are superior in terms of memory requirement and solution time. The iterative methods can be divided basically in two main groups, the stationary and the Krylov-Subspace methods. The stationary methods solve the equation system with an iteration matrix which is founded by splitting the system matrix. The Jacobi, Gauss-Seidel and Richardson method are examples of the stationary method. They generally converge far slower than the Krylov subspace methods, however they are effectively used as a preconditioner for them. The Krylov subspace iterative methods solve the equation system by repetitive multiplication of the system matrix to a vector thereby generating Krylov subspaces. They are considered the most important iterative solution technique for large and sparse linear systems [Saa03]. The most outstanding Krylov subspace methods are the generalized minimal residual method and the bi-conjugate gradient stabilized method beside the conjugate gradient method (CG) which is implemented in this work. There are many textbooks on the topic of iterative solution methods, here only a few are referred: [Saa03; Hac91; Mei05; Str16]. The conjugate gradient method was developed from [HS52]. It is nowadays mostly used for solving large and sparse linear equation system $\mathbf{K}\mathbf{u} = \mathbf{f}$ (3.40) with a symmetric positive definite matrix \mathbf{K} because of its efficiency, which only showed with the development of the first computers.

The convergence speed is highly influenced by a good preconditioning of the system matrix. A good preconditioner has to be cheap to construct and apply as well as to reduce the condition number of the preconditioned system. There are manifold preconditioners for different problems [Ben02]. But most preconditioners are obtained through the composed system matrix \mathbf{K} , which is not provided by the applied matrix-free methods (Ch. 5). In [GD01] the problematic application of preconditioners is shown in the context of the matrix-free framework with parallelization. The simplest preconditioner available

for matrix-free computations is the Jacobi or diagonal preconditioner. In [Ima+12] different diagonal preconditioner and corresponding weighting factors in terms of the minimal and maximal eigenvalue are evaluated for different problems solved with Krylov subspace methods and a solution to automatically ensure convergence is proposed. In matrix-free computations the computation of the weighting factor can only depend on the largest eigenvalue directly, so a weighting factor in function of the largest eigenvalue is proposed in [Arb+07] and in [Sch13]. Generated through reordering the displacement vector another matrix-free block-diagonal preconditioner is proposed in [ARS06].

The most promising preconditioner for matrix-free solution algorithms is the multi-grid method which will be described separately in Ch. 6. It also smooths slow frequency errors efficiently and overcomes therewith the limitation of stationary iterative methods such as Jacobi.

4.2 Error estimation and convergence assessment

Finite element discretization in linear elastic solid mechanics generally yields a symmetric and positive definite matrix \mathbf{K} (Sec. 3.3). The following equations are restrictively valid for a linear equation system with a symmetric and positive definite matrix ($\mathbf{K} = \mathbf{K}^\top$ and $\mathbf{v}^\top \mathbf{K} \mathbf{v} > 0 \forall \mathbf{v} \in \mathbb{R}^n; \mathbf{v} \neq \mathbf{0}$).

4.2.1 Error estimation

The purpose is to solve a large set of linear equations (3.40):

$$\mathbf{K} \mathbf{u} = \mathbf{f}$$

with the known right-hand side $\mathbf{f} \in \mathbb{R}^n$, the global stiffness matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ (symmetric and positive definite) and the unknown exact solution of this system $\mathbf{u} \in \mathbb{R}^n$. $\mathbf{v} \in \mathbb{R}^n$ denotes a computed approximation of the exact solution \mathbf{u} .

The algebraic error is then defined by

$$\mathbf{e} = \mathbf{u} - \mathbf{v} \tag{4.1}$$

and can be measured by any vector norm. The standard vector norm for an arbitrary vector \mathbf{x} through this work is the quadratic or Euclidean norm

$$\|\mathbf{x}\| = \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}. \tag{4.2}$$

If there is no further clarification, this is the considered error norm throughout this work.

The error is only available, when the solution is known, which is not the case. An accessible measurement is therefore the residual norm $\|\mathbf{r}\|$ from

$$\|\mathbf{r}\| = \|\mathbf{f} - \mathbf{K}\mathbf{v}\| \quad (4.3)$$

which measures the quantity of the failure of the approximation \mathbf{v} solving the original equation (3.40). For further clarification the residual equation can be evolved using the equations (3.40, 4.1 and 4.3)

$$\begin{aligned} \mathbf{K}\mathbf{u} - \mathbf{K}\mathbf{v} &= \mathbf{f} - \mathbf{K}\mathbf{v} \\ \mathbf{K}\mathbf{e} &= \mathbf{r}. \end{aligned} \quad (4.4)$$

The relative error in the residual is

$$\frac{\|\mathbf{r}\|}{\|\mathbf{r}_0\|} = \frac{\|\mathbf{f} - \mathbf{K}\mathbf{v}\|}{\|\mathbf{f} - \mathbf{K}\mathbf{v}_0\|} \quad (4.5)$$

with arbitrary start vector $\mathbf{v}_0 \in \Re^n$.

4.2.2 Convergence assessment

The convergence behavior of the solution process of a given linear equation system (3.40) is closely coupled to its eigenvalues: The scalar constant $\lambda \in \Re$ is an eigenvalue of $\mathbf{K} \in \Re^{n \times n}$ if a vector $\mathbf{x} \neq \mathbf{0}$ exists for which applies

$$\mathbf{K}\mathbf{x} = \lambda\mathbf{x}. \quad (4.6)$$

The eigenvalues are the solution of the eigenvalue problem

$$(\mathbf{K} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \quad : \mathbf{x} \neq \mathbf{0} \quad (4.7)$$

for a symmetric and positive definite system matrix \mathbf{K} and with the unit matrix \mathbf{I} which leads to the characteristic polynomial equation of the degree n

$$p_n(\lambda) = \det(\mathbf{K} - \lambda\mathbf{I}) = 0. \quad (4.8)$$

The spectrum of the matrix \mathbf{K} is the set of the eigenvalues

$$\sigma_K(\mathbf{K}) = \{\lambda_1, \lambda_2, \dots, \lambda_{n-1}, \lambda_n\} \quad (4.9)$$

and the spectral radius of \mathbf{K} is

$$\rho(\mathbf{K}) = \max\{\lambda_1, \lambda_2, \dots, \lambda_{n-1}, \lambda_n\} : \lambda \in \sigma_K(\mathbf{K}). \quad (4.10)$$

An important property of a system matrix to measure the accuracy of the approximate solution is the condition number κ

$$\kappa = \|\mathbf{K}\| \|\mathbf{K}^{-1}\|. \quad (4.11)$$

The condition number is used to assess the convergence velocity and quality of iterative methods. For a small condition number the assumption that a small residuum is equivalent to a small error holds true whereas with a large condition number the convergence behavior is in most cases unpredictable.

$$\frac{\|\mathbf{e}_k\|}{\|\mathbf{e}_0\|} \leq \kappa \frac{\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|}. \quad (4.12)$$

The matrix norm required for the convergence assessment can be an induced matrix norm

$$\|\mathbf{K}\|_i = \sup \frac{\|\mathbf{K}\mathbf{x}\|_i}{\|\mathbf{x}\|_i} \quad (4.13)$$

e.g. the maximum column sum

$$\|\mathbf{K}\|_1 = \max_{j=1..n} \sum_{i=1}^n |K_{ij}| \quad (4.14)$$

or the maximum row sum

$$\|\mathbf{K}\|_\infty = \max_{i=1..n} \sum_{j=1}^n |K_{ij}|. \quad (4.15)$$

The quadratic vector norm induced matrix norm, which is also called spectral norm can be estimated by

$$\|\mathbf{K}\|_2 = \max \frac{\|\mathbf{K}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq \sqrt{\sum_{i=1}^n \sum_{j=1}^m K_{ij}^2} = \|\mathbf{K}\|_F \quad (4.16)$$

where $\|\mathbf{K}\|_F$ is the Frobenius norm a p-norm treating the $n \times n$ matrix as vector of size nn and is not an induced matrix norm.

For symmetric and real matrices the spectral norm corresponds to the spectral radius

$$\|\mathbf{K}\|_2 = \rho(\mathbf{K}). \quad (4.17)$$

The condition number (4.11) induced by the spectral norm (4.16) is equal to the ratio of the largest λ_1 to the smallest eigenvalue λ_n of the symmetric and positive definite matrix \mathbf{K}

$$\kappa = \frac{\lambda_1}{\lambda_n}. \quad (4.18)$$

For the derivations and the proofs of the stated definitions in this section it is referred to the standard linear algebra textbooks such as [Mei05; Str16].

4.3 Stationary methods

The stationary methods calculate an estimation of the solution (3.40) based on the estimation of the previous steps. For this purpose the matrix \mathbf{K} is split such as

$$\mathbf{K} = \mathbf{A} + (\mathbf{K} - \mathbf{A}), \quad (4.19)$$

so the equivalent system to (3.40) is

$$\mathbf{A}\mathbf{u} = (\mathbf{A} - \mathbf{K})\mathbf{u} + \mathbf{f} \quad (4.20)$$

and with \mathbf{A} regular

$$\mathbf{u} = \mathbf{A}^{-1}(\mathbf{A} - \mathbf{K})\mathbf{u} + \mathbf{A}^{-1}\mathbf{f}. \quad (4.21)$$

Stationary methods are also referred as relaxation methods. The splitting matrix should be easy to construct and to multiply and a good approximation of the matrix \mathbf{K} . There are different methods of splitting the matrix, the most common and simple is the Jacobi method.

4.3.1 Jacobi method

The Jacobi method is cheap in terms of computational costs which is mostly coupled with a slow convergence behavior compared to more expensive methods. Nevertheless the Jacobi method is also easily applied to matrix-free methods which is generally not true for most methods and furthermore it is highly parallelizable.

The Jacobi method uses the diagonal matrix $\mathbf{D} = \text{diag}\{K_{11}, K_{22}, \dots, K_{nn}\}$ as a splitting matrix. The linear equation system (3.40)

$$\mathbf{K}\mathbf{u} = \mathbf{f}$$

is transformed to

$$\begin{aligned} \mathbf{u}^{k+1} &= \mathbf{u}^k + \mathbf{D}^{-1}(\mathbf{f} - \mathbf{K}\mathbf{u}^k) \\ \mathbf{u}^{k+1} &= \mathbf{M}\mathbf{u}^k + \mathbf{D}^{-1}\mathbf{f} \end{aligned} \quad (4.22)$$

where $\mathbf{M} = \mathbf{D}^{-1}(\mathbf{D} - \mathbf{K})$ is the iteration matrix and an arbitrary vector $\mathbf{u}^0 \in \mathbb{R}^n$ is the start vector. If the spectral radius satisfies $\rho(\mathbf{M}) < 1$ the Jacobi method converges. The convergence behavior improves when the spectral radius is $\rho(\mathbf{M}) \ll 1$. An exception is

the diagonal matrix as system matrix which can be solved in one iteration step (neglecting rounding errors) independently of the spectral radius.

The corresponding component notation of the Jacobi method is

$$u_i^{k+1} = \frac{1}{K_{ii}} \left[f_i - \sum_{j=1, j \neq i}^n K_{ij} u_j^k \right]. \quad (4.23)$$

4.3.2 Jacobi relaxation method

In the Jacobi relaxation method the Jacobi method is varied through a damping or relaxation factor ω :

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \omega \mathbf{D}^{-1}(\mathbf{f} - \mathbf{K}\mathbf{u}^k) \quad (4.24)$$

The spectral radius of the iteration matrix $\mathbf{M} = \mathbf{I} - \omega \mathbf{D}^{-1} \mathbf{K}$ with $\rho(\mathbf{M}) < 1$ is minimized when the optimal weighting factor [Mei05] is given by

$$\omega_{opt} = \arg \min_{\omega \in \mathbb{R}^+} \rho(\mathbf{M}(\omega)) = \frac{2}{2 - \lambda_1 - \lambda_n} \quad (4.25)$$

with λ_1 and λ_n as the maximal and the minimal eigenvalue respectively. Therewith the convergence rate is also increased for most equation systems.

The weighted Jacobi method can also be used to damp high frequency errors. The spectral radius is not necessarily decreased, but the higher frequencies are more efficiently damped. This is especially important if the weighted Jacobi method is used as damper for the multigrid method (Ch. 6). For a matrix-free multigrid method in [Arb+07] and [Sch13] a weighting factor in function of the largest eigenvalue of $\mathbf{D}^{-1} \mathbf{K}$

$$\omega = 0.75 / \lambda_{max}(\mathbf{D}^{-1} \mathbf{K}) \quad \text{and} \quad \omega = 1 / \lambda_{max}(\mathbf{D}^{-1} \mathbf{K})$$

respectively is proposed. In [Häf07] a heuristic weighting factor of $\omega = 0.8$ is used. The calculation of the maximal eigenvalue or of the equivalent spectral norm is omitted possibly risking a less effective smoothing. In this work the heuristically chosen weighting factor for the smoothing step in the multigrid method is $\omega = \frac{2}{3}$.

4.4 Conjugate gradients method

The conjugate gradient method is a well-known subtype of the Krylov subspace methods, which solves large and sparse linear equation system $\mathbf{K}\mathbf{u} = \mathbf{f}$ (3.40) with a symmetric positive definite matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ efficiently.

The quadratic function

$$U(\mathbf{u}) = \frac{1}{2} \mathbf{u}^\top \mathbf{K} \mathbf{u} - \mathbf{u}^\top \mathbf{f} \quad (4.26)$$

is minimized if

$$\hat{\mathbf{u}} = \arg \min_{\mathbf{u} \in \mathbb{R}^n} U(\mathbf{u}) \quad (4.27)$$

and

$$\mathbf{K}\hat{\mathbf{u}} = \mathbf{f}. \quad (4.28)$$

The minimum of the function U is found successively following special types of directions.

In the CG method the residual vectors are used to generate search directions. In each iteration step the matrix-vector multiplication spans a new Krylov subspace where the new search direction is \mathbf{K} -orthogonal and the new residual is orthogonal to all previous search directions and gradients. The CG method combines the positive properties of the method of steepest descent (problem orientated) and conjugate directions (optimized). The method converges theoretically to the exact solution after n steps and with considering round-up errors for the numerical solution the iteration is stopped after reaching a sufficient small tolerance ϵ which is generally reached for a much smaller number of iteration steps.

For a system of linear equations $\mathbf{K}\mathbf{u} = \mathbf{f}$ (3.40) where is \mathbf{u} the unknown displacement vector, \mathbf{f} a known external force vector and \mathbf{K} a known, square, symmetric, positive-definite matrix, the solution algorithm of the conjugate gradient method is shown in algorithm 4.1.

Algorithm 4.1 CG

- 1: Choose a start vector $\mathbf{u} \in \mathbb{R}^n$
 - 2: $\mathbf{d} := \mathbf{r} := \mathbf{r}_0 := \mathbf{f} - \mathbf{K}\mathbf{u}$
 - 3: $\alpha_1 := \mathbf{r}^\top \mathbf{r}$
 - 4: $\epsilon_0 := \alpha_1$
 - 5: **while** $\epsilon > tol^2 \epsilon_0$ **do**
 - 6: $\mathbf{h} := \mathbf{K}\mathbf{d}$
 - 7: $\alpha := \frac{\alpha_1}{\mathbf{d}^\top \mathbf{h}}$
 - 8: $\mathbf{u} := \mathbf{u} + \alpha \mathbf{d}$
 - 9: $\mathbf{r} := \mathbf{r} - \alpha \mathbf{h}$
 - 10: $\beta_1 := \mathbf{r}^\top \mathbf{r}$
 - 11: $\beta := \frac{\beta_1}{\alpha_1}$
 - 12: $\mathbf{d} := \mathbf{r} + \beta \mathbf{d}$
 - 13: $\epsilon := \alpha_1$
 - 14: $\alpha_1 := \beta_1$
 - 15: **end while**
-

The conjugate gradient algorithm is further enhanced through preconditioning to accelerate the convergence process in the next section.

4.4.1 Preconditioned conjugate gradient method

The convergence rate of the conjugate gradient method can be further accelerated through preconditioning. The original problem $\mathbf{K}\mathbf{u} = \mathbf{f}$ is transferred to

$$\mathbf{P}_L \mathbf{K} \mathbf{P}_R \hat{\mathbf{u}} = \hat{\mathbf{f}} \quad (4.29)$$

with $\hat{\mathbf{u}} = \mathbf{P}_R^{-1} \mathbf{u}$ and $\hat{\mathbf{f}} = \mathbf{P}_L \mathbf{f}$, in order that the condition number (Section 4.2) of the system matrix is improved

$$\kappa(\mathbf{P}_L \mathbf{K} \mathbf{P}_R) \ll \kappa(\mathbf{K}). \quad (4.30)$$

As the conjugate gradient method requires a symmetric and positive-definite system matrix the right preconditioned matrix \mathbf{P}_R has to be the transposed of left preconditioned matrix \mathbf{P}_L : $\mathbf{P}_R = \mathbf{P}_L^\top$ leading to the symmetric and positive definite transformed equation system

$$\mathbf{P}_L \mathbf{K} \mathbf{P}_L^\top \hat{\mathbf{u}} = \hat{\mathbf{f}} \quad (4.31)$$

with $\hat{\mathbf{u}} = \mathbf{P}_L^{-\top} \mathbf{u}$ and $\hat{\mathbf{f}} = \mathbf{P}_L \mathbf{f}$.

This equation system can be solved with the conjugate gradient method. For the derivation of the PCG method in function of the original variables a matrix \mathbf{P} that fulfills $\mathbf{P} = \mathbf{P}_L^\top \mathbf{P}_L$ is introduced and $\hat{\mathbf{r}} = \mathbf{P}_L \mathbf{r}$, $\hat{\mathbf{d}} = \mathbf{P}_L^{-\top} \mathbf{d}$ and $\hat{\mathbf{u}} = \mathbf{P}_L^{-\top} \mathbf{u}$ are back substituted. For a more detailed derivation it is referred to the corresponding textbooks (e.g. [Mei05]). The algorithm of the general preconditioned conjugate gradient method (PCG) is represented in Alg. 4.2.

An efficient preconditioner has to be

- a good approximation of the inverse matrix \mathbf{K}^{-1} ,
- easy to construct
- cheap to apply.

Most preconditioners require the system matrix as an assembled matrix which is not available in the matrix-free methods (Ch. 5). Only the implemented and applied preconditioners suitable for matrix-free methods are described in this work: in the following the simplest form of a preconditioner the weighted Jacobi preconditioner and in Sec. 6.3 of Ch. 6 the most efficient preconditioner for regular grid models - the multigrid method.

Jacobi preconditioner

The most straightforward preconditioning matrix \mathbf{P} applicable to the linear equation system $\mathbf{K}\mathbf{u} = \mathbf{f}$ (3.40) is the Jacobi or diagonal preconditioner $\mathbf{P} = \mathbf{D}^{-1}$ which uses

Algorithm 4.2 PCG

```

1: Choose a start vector  $\mathbf{u} \in \mathbb{R}^n$ 
2:  $\mathbf{r} := \mathbf{r}_0 := \mathbf{f} - \mathbf{K}\mathbf{u}$ 
3:  $\mathbf{d} := \mathbf{z} := \mathbf{P}\mathbf{r}$ 
4:  $\alpha_1 := \mathbf{r}^\top \mathbf{z}$ 
5:  $\epsilon_0 := \mathbf{r}^\top \mathbf{r}$ 
6: while  $\epsilon > tol^2 \epsilon_0$  do
7:    $\mathbf{h} := \mathbf{K}\mathbf{d}$ 
8:    $\alpha := \frac{\alpha_1}{\mathbf{d}^\top \mathbf{h}}$ 
9:    $\mathbf{u} := \mathbf{u} + \alpha \mathbf{d}$ 
10:   $\mathbf{r} := \mathbf{r} - \alpha \mathbf{h}$ 
11:   $\mathbf{z} := \mathbf{P}\mathbf{r}$ 
12:   $\beta_1 := \mathbf{r}^\top \mathbf{z}$ 
13:   $\beta := \frac{\beta_1}{\alpha_1}$ 
14:   $\mathbf{d} := \mathbf{z} + \beta \mathbf{d}$ 
15:   $\alpha_1 := \beta_1$ 
16:   $\epsilon := \mathbf{r}^\top \mathbf{r}$ 
17: end while

```

the diagonal matrix \mathbf{D} of the system matrix \mathbf{K} :

$$\mathbf{D} = \text{diag}\{K_{11}, K_{22}, \dots, K_{nn}\}. \quad (4.32)$$

The diagonal matrix is easily computed and applied.

Even for the Jacobi-preconditioner different diagonal matrices are possible such as various block-diagonal. The above described diagonal preconditioner is referred as point-Jacobi. The diagonal preconditioner has low computational cost and high parallel efficiency which unfortunately comes with a slow convergence rate in most cases. An additional weighting factor can speedup the convergence rate or can be used for damping high frequency errors when the method is used as a smoother. The preconditioning matrix is then $\mathbf{P} = \omega \mathbf{D}^{-1}$ and is referred as damped or weighted Jacobi preconditioner.

4.4.2 Benchmark: Weighted Jacobi preconditioner

An cut-out with the size of 70 voxels per direction of the bone biopsy (*bone*) of a femoral neck (Fig. 2.1) with 302736 dofs is considered under an uni-axial displacement-driven load to show the effect of the Jacobi preconditioner with and without scaling through a weighting factor ω . The following boundary conditions are applied: $u_z = 10^{-2} z_{max}$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = z_{max}$ and $u_x = 0$ for $x = 0, y = \{0; y_{max}\}, z = \{0; z_{max}\}$; $u_y = 0$ for $x = \{0; y_{max}\}, y = 0, z = \{0; z_{max}\}$ and $u_z = 0$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = 0$. $\{\cdot\}_{max}$ is the maximal value of the model dimension in $\{\cdot\}$ -direction. The dimension of one voxel is based on the bone example $1,948 \cdot 10^{-2}$ mm in each

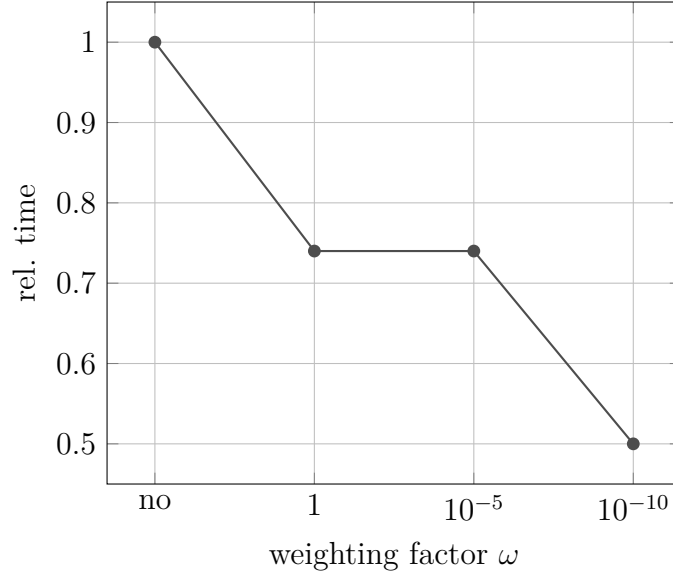


Figure 4.1: Rel. solution time of the weighted JCG with different weighting factors vs. the CG

direction. The linear elastic material parameter are $E = 100000N/mm^2$ and $\nu = 0,2$. The termination criterion used for the JCG is $\|\mathbf{r}\| \leq 10^{-6}\|\mathbf{r}_0\|$. The model is calculated without preconditioner (no) and with a weighting factor of $\omega = 1$, $\omega = 10^{-5}$ and $\omega = 10^{-10}$.

The easily applied Jacobi preconditioner reduces the computation time about 30% and the weighted Jacobi preconditioner with $\omega = 10^{-10}$ leads to a reduction of about 50% compared to the conjugate gradient method without preconditioner. For the eventual comparison to the multigrid preconditioner the Jacobi preconditioned conjugate gradient method (JCG) without weighting factor ($\omega = 1$) is considered.

Chapter 5

Matrix-free methods

An important scope of this work is the solution of the large sparse linear equation systems without actually assembling and storing the global system matrix and therewith save memory space. The regular Cartesian grid structure of the input data intrinsically provides concurrent voxel-based elements with repetitive connectivity information. The matrix-free method is an implicit part of each solution procedure in this work and is therefore described independently of the iterative methods which were introduced in Chapter 4. In this chapter two distinct matrix-free methods are presented where the global matrix-vector calculation is done on the local element or node level. Special issues of both methods are demonstrated and for different benchmark problems the results are shown and evaluated. Both alternatives will be critically reviewed with respect to implementation and performance issues. This part of the work was also presented in [KK12a].

5.1 State of the art

In large equation systems the global stiffness matrix gets very large and results in a memory overrun. Through their regular structure grid-based models provide the possibility for matrix-free solution techniques which store one basic element stiffness matrix and calculate the matrix-vector product locally. With the adaption from global to small local matrix-vector calculation the memory restriction is overcome through the drastic reduction of the memory demand and large models can be solved on modern desktop computers. The most time consuming step in each iteration algorithm, the matrix-vector product, is now computed without the assembling and saving of the global system matrix. In commercial software the matrix-free solution algorithm are not included and the advantages of regular voxel-based models cannot be applied [AEM06]. Matrix-free solution techniques were first used by [HLW83] and nowadays element-by-element methods are applied for many large scale problems e.g. for offshore structures [Cou+87], for seismic wave modeling [Liu+14], for a time-transient proliferation of cellular tissue [Zoh15] and for simulation of laser processing of particulate-functionalized materials [Zoh17]. For

image-based μ FE analysis of bone matrix-free approaches were first applied in [HK94], further advances were presented in [Rie+95; Rie+96; NCG05; Arb+07; FA11a] chronologically. Discretization and solution techniques such as nodal high order finite elements [Bro10], quasi-static frictional contact problems [HL99], GPU parallelization [MMH15; DGW11], for immersed continuum methods [Wan07], the adaption to a cell-based local scheme [KK12b] are based upon matrix-free implementation. For fluid mechanics a matrix-free procedure is applied to the implicit finite volume lattice Boltzmann method [Li17], to incompressible Navier-Stokes equations [FD16], to heterogeneous Stokes flow [MBL15] and to Galerkin formulations of compressible flows [Gao+17]. A multigrid solver for Poisson equations with a matrix-free implementation applied to FE and variants of the discontinuous Galerkin method is up to order of magnitude faster compared to matrix-based implementation due to vastly better performance of matrix-free operation compared to sparse matrix [KW16].

In [RC05] also a comparison between matrix-free methods is presented and it is found that for hexahedron the EBE is particularly suited and for unstructured grids the edge-by-edge method is superior to element-by-element solutions for viscous-plastic flows in [EMC06] as well as for solid mechanics in [Cou+01]. In [Rie+96] an improved row-by-row matrix-vector product as a variant of the element-by-element method is proposed.

5.2 Adaption of the matrix-vector product

Solving the large sparse linear equation system $\mathbf{K}\mathbf{u} = \mathbf{f}$ (3.40) globally is omitted with the application of matrix-free methods. Generally the equation is transformed for iterative methods (Ch. 4) such that

$$\mathbf{r} = \mathbf{f} - \mathbf{K}\mathbf{u}. \quad (5.1)$$

The matrix-vector product, which is the most time consuming part of the solution algorithm [RC05], is rewritten by introducing the vectors \mathbf{h} and \mathbf{d} :

$$\mathbf{h} = \mathbf{K}\mathbf{d}. \quad (5.2)$$

\mathbf{h} is a help vector for the inner step of the matrix-vector solution and \mathbf{d} is the search direction calculated appropriately the applied iterative method. The Eq. (5.2) is then computed with matrix-free methods. In the next sections two matrix-free methods are presented and then compared with respect to CPU time and memory demand considering an underlying Cartesian grid model.

5.3 Element-by-element method

One benefit of regular grid discretization with elements of the same size, orientation and shape is the simplified topology, which allows a substantially reduced data structure with minimal memory requirement. Furthermore, only one element stiffness matrix for each Poisson's ratio is provided through taking into account different materials by changing the Young's modulus as a scalar factor. In contrast to the element-by-element method presented in [RC05] where all element stiffness matrices are stored this reduces the required main memory substantially. Therewith huge models are feasible on modern desktop computers.

On this basis, the linear equation system is solved with local matrix-vector calculations, without calculation or storage of the global stiffness matrix (matrix-free) and this way the computational effort is reduced as well as the needed memory demand. The remaining local matrix-vector calculations are of the order of the numbers of dofs of one element. E.g. for a eight node hexahedral element it is a 24x24 matrix-vector calculation.

The matrix-vector operation (5.2) is changed in such a way that for each element the local element stiffness matrix is multiplied with the corresponding dofs of the element:

$$\mathbf{h} = \sum_e \mathbf{T}^\top E^e \mathbf{K}^0 \mathbf{T} \mathbf{d} \quad (5.3)$$

\mathbf{K}^0 is the local basis element stiffness based on a Young's modulus $E = 1$ which is multiplied with the Young's Modulus E^e of the respective element e and \mathbf{T} is the global-to-local transformation matrix. The global transformation matrix \mathbf{T} is not assembled and stored. It is replaced by a simple calculation function which is possible due to the regular grid structure. \mathbf{h} is calculated ones per iteration step with the algorithm 5.1: n_e are the numbers of elements and the index e corresponds to the element id. For each element the local dofs of the search direction vector \mathbf{d} are collected and then multiplied with the local stiffness matrix. Finally the local solution \mathbf{h}^e is assembled back in the global vector. The storing of nodes at the element is tested versus the calculation of the nodes through their location in the grid in Sec. 5.5.2.

The changes in the preconditioned conjugate gradient algorithm (PCG) implying the EBE matrix-vector operation are shown in Alg. 5.2.

Algorithm 5.1 Element-based matrix-vector product (EBE) MatVecEBE(\mathbf{h}, \mathbf{d})

- 1: Set $\mathbf{h} = \mathbf{0}$.
 - 2: **for** $e = 1 \rightarrow n_e$ **do**
 - 3: Collect $\mathbf{d}_e = \mathbf{T} \mathbf{d}$.
 - 4: Calculate $\mathbf{h}^e = E^e \mathbf{K}^0 \mathbf{d}_e$.
 - 5: Assemble $\mathbf{h} += \mathbf{T}^\top \mathbf{h}^e$.
 - 6: **end for**
-

Algorithm 5.2 EBE-PCG

```

1: Choose a start vector  $\mathbf{u} \in \mathbb{R}^n$ 
2:  $\mathbf{r} := \mathbf{r}_0 := \mathbf{f} - \sum_e \mathbf{T}^\top E^e \mathbf{K}^0 \mathbf{T} \mathbf{u}$ 
3:  $\mathbf{d} := \mathbf{z} := \mathbf{P} \mathbf{r}$ 
4:  $\alpha_1 := \mathbf{r}^\top \mathbf{z}$ 
5:  $\epsilon_0 := \mathbf{r}^\top \mathbf{r}$ 
6: while  $\epsilon > tol^2 \epsilon_0$  do
7:    $\mathbf{h} = \sum_e \mathbf{T}^\top E^e \mathbf{K}^0 \mathbf{T} \mathbf{d}$ 
8:    $\alpha := \frac{\alpha_1}{\mathbf{d}^\top \mathbf{h}}$ 
9:    $\mathbf{u} := \mathbf{u} + \alpha \mathbf{d}$ 
10:   $\mathbf{r} := \mathbf{r} - \alpha \mathbf{h}$ 
11:   $\mathbf{z} := \mathbf{P} \mathbf{r}$ 
12:   $\beta_1 := \mathbf{r}^\top \mathbf{z}$ 
13:   $\beta := \frac{\beta_1}{\alpha_1}$ 
14:   $\mathbf{d} := \mathbf{z} + \beta \mathbf{d}$ 
15:   $\alpha_1 := \beta_1$ 
16:   $\epsilon := \mathbf{r}^\top \mathbf{r}$ 
17: end while

```

5.4 Node-edge based method

$$\mathbf{h} = \sum_n \mathbf{T}^\top \sum_j \sum_k E^{jk} \mathbf{K}^{0j} \mathbf{T} \mathbf{d} \quad (5.4)$$

The node-edge based method developed for this work is a mixture of a node- and an edge-based approach, the algorithm is shown in Alg. 5.3. The loop is done over the nodes n and for each node a loop over all edges¹ j to the neighbor nodes (27 nodes) is executed, multiplying the edge-based part j of the stiffness matrix of each adjacent element k with the Young's modulus E^{jk} of the element and the degrees of freedom (dofs) of the corresponding neighbor node. The matrix-vector operation is a vector of 3 components (dofs of one node) multiplied by a 3x3-matrix. The corresponding transformation matrix would be a 3x3n matrix if assembled.

Each edge belongs to one element, so that a basis stiffness matrix can be taken and multiplied with the Young's modulus of the underlying element. Consequently their number of edges between the same two nodes correspond to the number of adjacent elements. The separate handling enables different materials in neighbor elements considered only by a scalar factor. The number of edges is then each edge multiplied by the number of neighbor elements: $8 \times 1 + 12 \times 2 + 6 \times 4 + 1 \times 8 = 64$. For non existing neighbors this

¹Here an edge is the connection of one node to another node similarly to ansatz functions and not only corresponding element edges.

method multiplies a zero value for E^{jk} .

Algorithm 5.3 Node-edge based matrix-vector product (NE)

```

1: for  $n = 1 \rightarrow n$  do
2:     for  $j = 1 \rightarrow 64$  do
3:         Collect  $\mathbf{d}^j = \mathbf{T}\mathbf{d}$ 
4:         Calculate  $\mathbf{h}^{n+} = \sum_k E^{jk} \mathbf{K}^{0j} \mathbf{d}^j$ 
5:     end for
6:     Assemble  $\mathbf{h} = \mathbf{T}^\top \mathbf{h}^n$ 
7: end for

```

Fig. 5.1 visualizes a patch of eight neighbor elements adjacent to the node $n = 13$. The neighbor nodes are numbered for simplification in the way the algorithm loops over the nodes. So between the neighbor nodes 13 and 1 the two edges are located at the same place, because of the two associated elements. Applied to the element the position of the edge is different and therewith the part of the stiffness matrix which has to be considered. The separate handling enables different materials in neighbor elements without saving the assembled part of the stiffness matrix for all nodes.

The matrix size is 3×3 for a node with three degrees of freedom. The basic coefficient matrix for each edge of each associated element is stored, in 3D with a linear ansatz function there are 64 edges, so 64 local 3×3 stiffness matrices are stored. The data locality of the ordered node numbering remains at the expense of a double loop and in double saving of the edge stiffness¹ to keep the nodes independent. In the node-edge based solution algorithm each node is solved completely separate from the other, which is especially well suited for parallelization.

5.5 Examples: Academic and bone models

Two matrix-free computation methods for iterative solutions of sparse linear systems, an element-by-element and a node-edge based method, are compared with respect to computation time and memory demand. Both alternatives are based on (and taking the advantages of) a regular three dimensional grid structure (with row-major data storage in these examples) as a special case of finite element discretization. The examples are solved with the weighted (10^{-10}) Jacobi preconditioned conjugate gradient method.

The methods developed in this work are intended to be used on modern desktop computers with multi-core CPUs² or moderate large workstation clusters. So the number of CPUs and available memory remains relative small compared to the model size and in consequence the restriction of memory is mandatory for the calculation of very large

¹The edge stiffness matrix from node A to node B is apparently the same as from B to A although it is saved at both nodes.

²The in the examples used workstation configuration is a 2nd generation Intel core I7-2600 processor (8M Cache, up to 3.80 GHz) and 8GB (2x4GB) RAM.

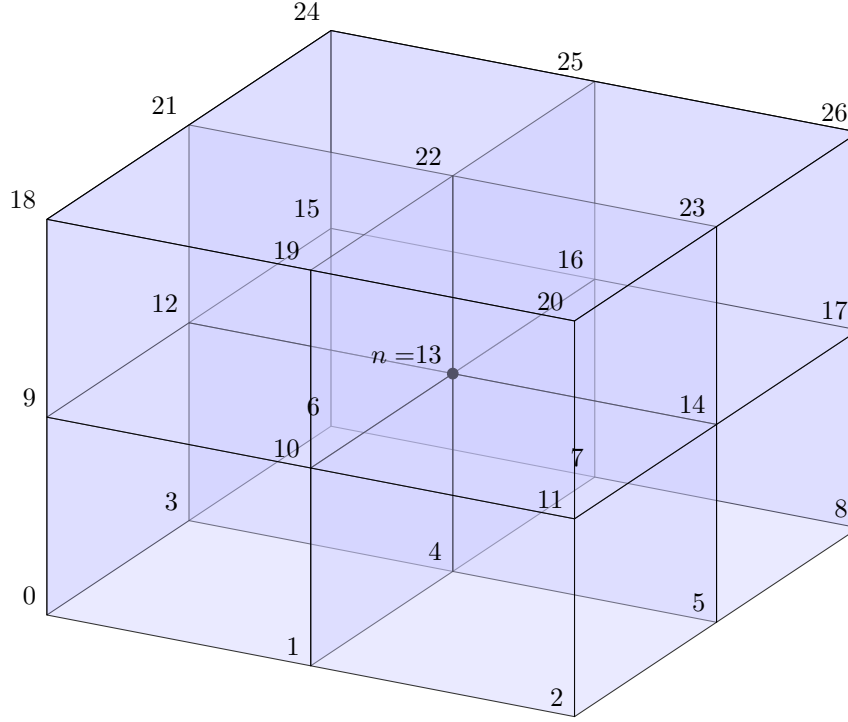


Figure 5.1: 3D patch of the eight elements belonging to the node $n = 13$

models. Hence the overall goal of the proposed approach is to save memory before computing time. So some extra computations are accepted, as long as there is a tradeoff with respect to the main memory consumption. To illustrate these influences an example of the element-by-element method with two different ways to handle associate node information is shown (Sec. 5.5.2).

In order to validate the proposed approach two academic examples and one bone biopsy are loaded with an uni-axial displacement-driven traction in z-direction $u_z = 10^{-2} z_{max}$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = z_{max}$ and constrained by displacement boundary conditions: $u_x = 0$ for $x = 0, y = \{0; y_{max}\}, z = \{0; z_{max}\}$; $u_y = 0$ for $x = \{0; y_{max}\}, y = 0, z = \{0; z_{max}\}$ and $u_z = 0$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = 0$. $\{\cdot\}_{max}$ is the maximal value of the model dimension in $\{\cdot\}$ -direction. The dimension of one voxel is based on the bone example $1,948 \cdot 10^{-2}$ mm in each direction. The linear elastic material parameter are for all examples in the work $E = 100000 N/mm^2$ and $\nu = 0,2$. The examples are comparable by the numbers of voxels in each spatial direction, the investigated model sizes are cubes of 70, 100 and 200 voxel per direction. The first example is a simple uniform cube (*block*). The number of dofs for the 70, 100 and 200 example is 1,073,733 / 3,026,430 and 24,361,803 respectively. The next academic example is an uniform cube with a cubical hole (*hole* model, left figure of Fig. 5.2) with 933,765 / 2,715,903 and 21,361,803 dofs for the different resolutions. For the academic examples (*hole* and *block*) between 100 and 200 voxels also intermediate resolutions are studied.

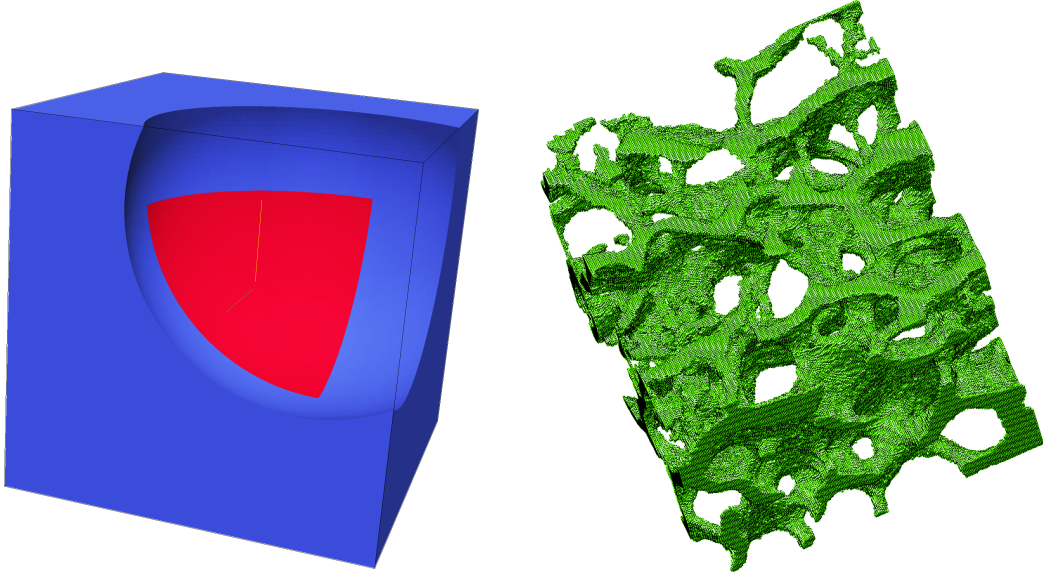


Figure 5.2: Cube with cubical hole and bone of femoral neck biopsy [PB03] examples

The bone biopsy (*bone* model, right figure of Fig. 5.2) as real example with complex microstructure is considered for 70, 100 and 200 voxels per direction with 302,736 / 684,582 and 4,936,095 dofs respectively and in contrary to the other two examples it is not the same bone extract, but the resolution remains constant while the size of the sample changes. The bone data is taken of the femoral neck [PB03]. The data is obtained by an X-ray μ CT scanning. A binary region of interest data set with about 200 pixels per 4 mm in each dimension is the input of our basis model. The smaller examples are cut-outs from the base model.

The Fig. 5.3 shows the relation of the dofs to the dimension of the model for the three examples.

5.5.1 Benchmark: Matrix-free versus ANSYS solution

To compare the efficiency of the proposed methods with standard FE approaches the large *hole* model with $\text{dim}=200$ and the 100 voxels per dimension *bone* model are compared with results from the commercial finite element software ANSYS¹ solved with the ANSYS PCG solver. The symmetric and sparse structure of the stiffness matrix is automatically considered, only the symmetric non-zero part of the stiffness matrix is kept in memory as compressed sparse row (CSR) format. The results are listed in Tab. 5.1.

Apparently the matrix-free method is not as efficient as the ANSYS solution technique when comparing the computing time but much more efficient in main memory consumption. Considering modern desktop computers the amount of provided memory

¹ANSYS Academic Research, Release 12.1, ANSYS Inc., www.ansys.com

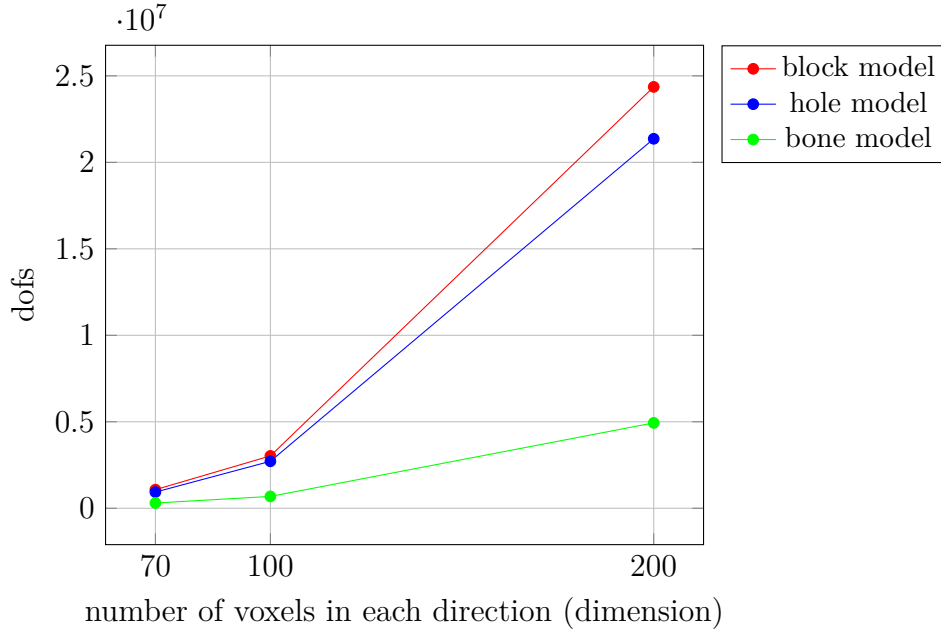


Figure 5.3: Numbers of dofs of the uniform cube (*block*), the void cube in cube (*hole*) and the bone model (*bone*) versus model dimension

restricts the model size. The *hole* model example needs more than 16 GB main memory using ANSYS which is in the limit range of current desktop computers. The iterative solution process is further improved in the progress of this work through efficient preconditioning with the multigrid method (Ch.6). Furthermore regarding the time of model construction (pre-processing) the presented method is faster as the commercial software ANSYS. The time demand of the pre-processing is not shown here, because in ANSYS it is done through a standard input file reading in each element separately which is extremely slow. This should be improved for a fair comparison. Nevertheless it has to be stated that the pre-processing for the here proposed methods is quite fast.

	<i>Hole</i> (dim=200)		<i>Bone</i> (dim=100)	
	solution time (min)	memory (GB)	solution time (min)	memory (MB)
NE	33.25 (90.9%)	4.8 (400%)	4.73 (103%)	161 (355%)
EBE	36.58 (100.0%)	1.2 (100%)	4.6 (100%)	45 (100%)
ANSYS	15.58 (42.6%)	16.0 (1333%)	1.7 (37%)	4484 (9964%)

Table 5.1: Results of ANSYS vs. NE and EBE solution in terms of computing time and main memory demand

5.5.2 Benchmark: Storage versus calculation of associate node information in the EBE method

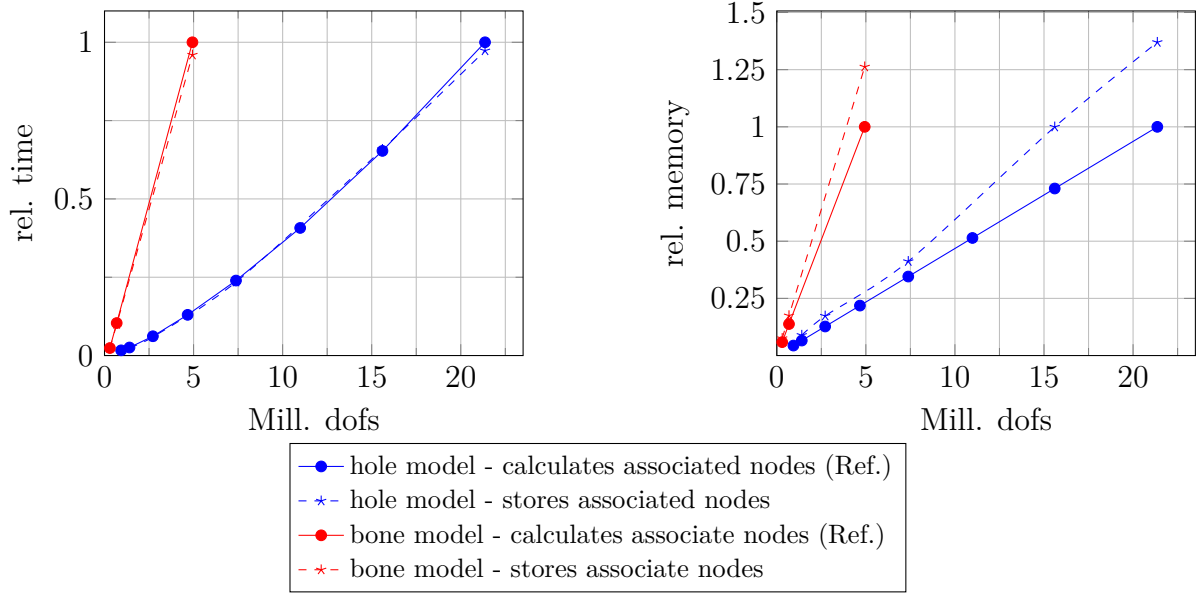


Figure 5.4: Comparison of storage or calculation of associate node information in EBE, results are relative to the greatest *hole* or *bone* model with associate node information calculation respectively

In this subsection the element-by-element method which calculates the associate node numbers of an element in each iteration (solid lines) is compared to an EBE which saves the associate node information for each element (dashed lines). As Fig. 5.4 shows the time gain is negligible while the amount of memory increases remarkable by more than 25% when saving the additional data. Henceforth both matrix-free methods (EBE and NE) are implemented and applied in versions with minimal memory demand while all necessary model data is calculated on the fly during the solution process.

5.5.3 Benchmark: EBE versus NE method

The *block*, *hole* and *bone* model examples are calculated with different sizes and the results are compared with respect to time and memory demand. The results for the different models are named and visualized in the following ways: void cube in cube model - *hole* (blue —●—), uniform cube model - *block* (red —●—) and bone model - *bone* (green —●—). The line style is solid (—●—) for the EBE and dashed for the NE method (—*—).

Fig. 5.5 (a) shows the computing time for the solution versus degrees of freedom (dofs) relative to the EBE solution. For the *block* and *hole* example the NE method is slightly faster than the EBE method whereas for the *bone* model example it is reversed. So for

complicated geometries the EBE is slightly superior to the NE. Furthermore taking into account the memory demand (Fig. 5.5 (b)) the EBE method is clearly more suited for the Cartesian grid models, because the memory demand of the NE in comparison to the EBE method increases with a factor of 4 for the academic models and a factor of 3.5 for the bone model.

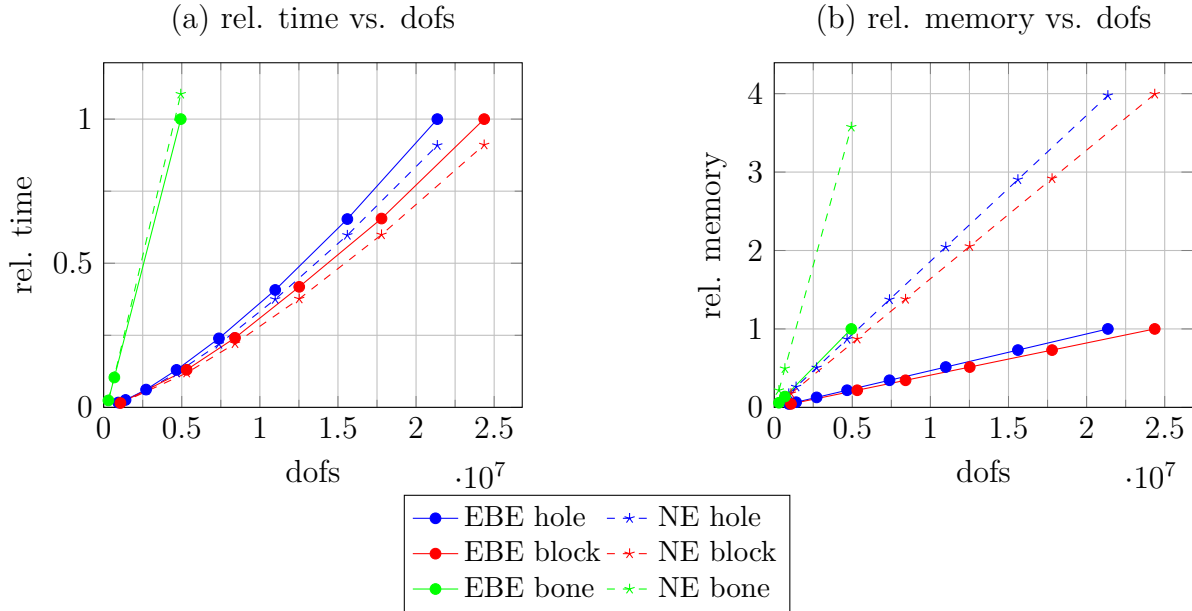


Figure 5.5: Comparison of solution time and memory demand relative to the respective EBE solution versus degrees of freedom (dofs)

5.5.4 Comparison of the solution time of the three models

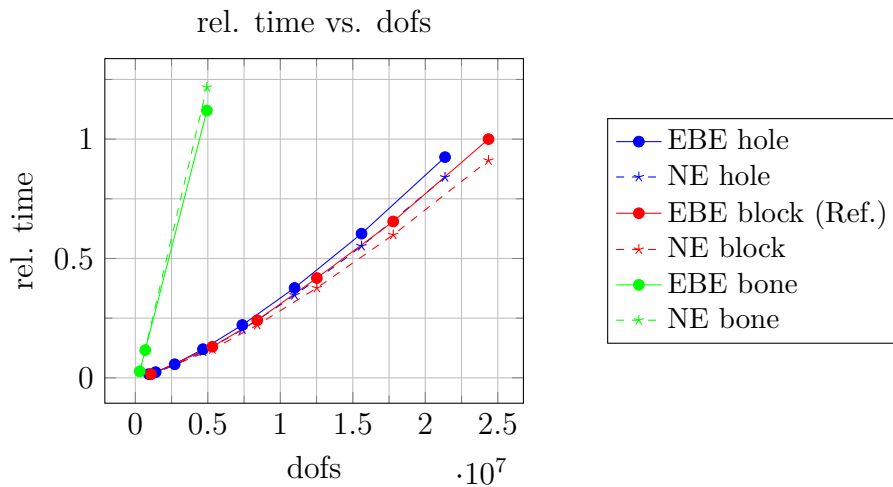


Figure 5.6: Solution time relative to the EBE block model (dim=200) versus degrees of freedom (dofs)

Fig. 5.6 shows the solution time for all models relative to the block model with a resolution of 200 voxels per dimension. The bone model examples need even more time than the two academic examples although the bone model consists of considerably less elements and therewith dofs. So remembering that the same cube is considered with simple (block) as well as a complex (bone) structure and that the solution algorithm is iterative, it is reasonable, that a model with homogeneous material distribution converges faster than a heterogeneous one like the trabecular bone example. The applied Jacobi preconditioned Conjugate Gradient method reveals slow convergence behavior for high frequency errors caused by heterogeneous models. This effect can be overcome by suitable preconditioning e.g. through the proposed multigrid method (Ch. 6).

5.5.5 Conclusion

In this chapter the matrix-free method has been introduced and two methods to solve the matrix-vector product locally have been presented. As shown in the Subsection 5.5.3 for the uniform cube and cube in cube model the EBE method is slightly slower for the academic models but needs considerable less memory. For the bone model the EBE method is faster whereas the memory amount is likewise lower. So generally the EBE method is more advantageous, because the main memory need is significantly smaller which is an important aspect when solving large systems on desktop computers. Nevertheless the NE method stays in the scope of interest for unstructured meshes [Cou+01]. Furthermore the additional computing time for the calculation of associate node information which remains small whereas the memory demand decreases by more than 25% compared to the stored data example. As minimizing the memory demand is the most critical target variable the methods are implemented in such a way that all required information like nodes at the element, neighbor elements etc. are computed in the solution process.

Chapter 6

Multigrid methods

Multigrid methods provide a fast and robust solution algorithm for linear as well as non-linear or time-dependent equation systems. The fine grid equation system is approximated on a coarser grid recursively which reduces the problem size and is therewith computational cheaper. In the context of this work the multigrid method is implemented as a preconditioner to the conjugate gradient method (MGCG) for the matrix-free voxel-based analysis however also works as a standalone solver.

6.1 State of the art

The multigrid method as one defined method does not exist, instead it is a strategy of solving computational problems [Bri87]. The earliest multigrid methods were presented by Federenko and Bachwalow as relaxation method [Fed62; Bac66] and then by Brandt as multilevel method [Bra73; Bra77].

The algorithm approximates the error of the fine grid discretization on a coarser grid which is due to the smaller equation system numerically cheaper. Applied recursively to a hierarchy of coarser grids it leads to a multigrid algorithm which scales linear. The multigrid is a fast iterative solver for various differential equation e.g. Laplace [SB10], Poisson [KW16; Sti16], Navier-Stokes [HMS08], Maxwell [HW16]. It is applicable to arbitrary regions and boundary conditions.

For Cartesian finite element discretization the application of the multigrid concept is straightforward. Beside this geometrical approach the algebraic multigrid modifies the linear equation system directly without geometrical consideration and thus is used for complex geometries [Aug+16] and problems without geometrical background, e.g. time-dependent analysis. Some general textbooks and tutorials on the multigrid method which provide basic and detailed information are [Bra93; Bri87; Hac85; Wes92].

In this work the multigrid method is implemented for matrix-free voxel-based analysis as a preconditioner for the iterative solution method of conjugate gradients (MGCG) although the multigrid can also be applied as standalone solver [Bol+03; BD85]. The

results of both variants are presented. The superior convergence rate of the multigrid preconditioned conjugated gradient method (MGCG) over the multigrid solver found in [Ket82; Tat93] for problems with large coefficient jumps in-between materials are validated in the Sec. 6.5. Similar results have been presented in [Bra86] for computations with global matrices even with the possibility of pre-computing the MG preconditioner as a matrix. The MGCG is applied to grid-based heterogeneous models in [Häf07; HK06]. In [FA12] the MGCG is deployed in a parallel implementation of μ FE analysis with voxel-based discretization. Without implying completeness, the multigrid is extended in further developments to composite finite elements for solving image-based analysis [Lie+09], to octree-based discretization [SB10], to iso-geometrical analysis [HZ15] and to unstructured grids with regular elements [GWX16].

6.2 Introduction to the multigrid methods

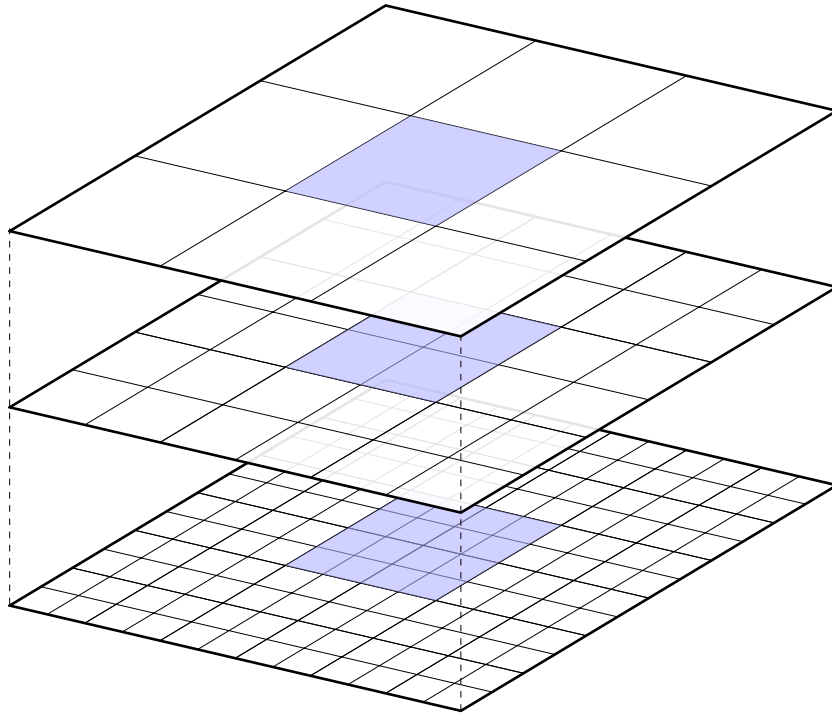


Figure 6.1: Multigrid construction of three grid levels in 2D

The main feature of the multigrid (MG) methods is name-giving: multi grids (Fig. 6.1). The problem is transferred to coarser successive levels, solved and the solution is transferred back to the original level. The coarser grid solution is a good approximation of the fine grid solution of a physical problem with considerably less computational effort. The multiple possibilities of the transfer of the grids is one main point of the adaptivity of the multigrid scheme. The most common grid cycles, V-, W- and saw-tooth-cycle, are shown in Fig. 6.2.



Figure 6.2: Different types of multigrid cycles: V-cycle, variant of W-cycle and saw-tooth-cycle respective

Traditionally the grid level number for the finest grid is the highest number and the coarsest the number 0. In this work we consider models derived from digital images with high resolution, so the starting level is the finest grid and hence will be named as zero level $l = 0$.

Considering Cartesian grid models with dim as the largest number of voxel elements per edge the maximum number of grid levels l_{max} is

$$dim = 2^{l_{max}}. \quad (6.1)$$

The mesh width h is doubled for each coarser level

$$h_{l+1} = 2h_l. \quad (6.2)$$

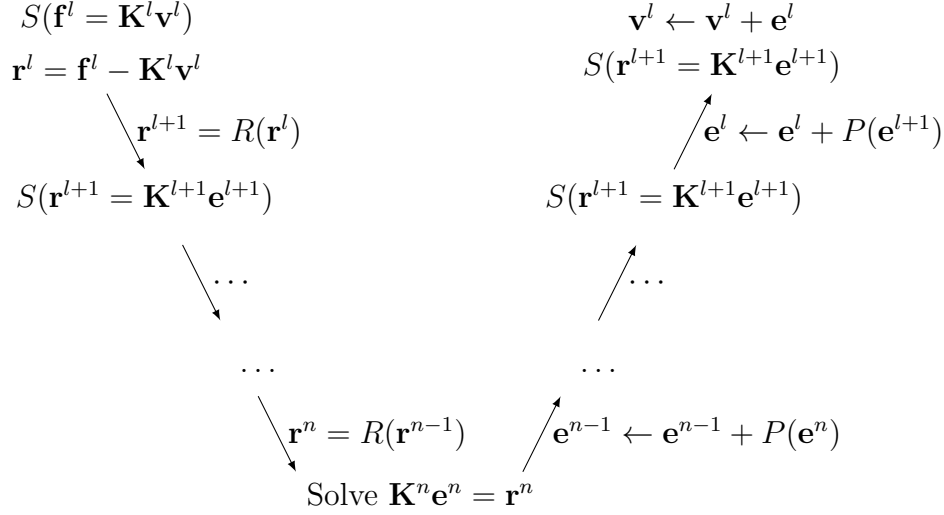
From the finest level l_0 to an arbitrary level l the mesh width is obtained by

$$h_l = 2^l h_0. \quad (6.3)$$

On each level the following steps have to be executed:

- **Smoothing S :** Smooth the high frequency errors in the solution.
- **Restriction R :** The residual is transferred to the coarser grid.
- **Prolongation P :** The error solution is transferred to the finer grid.
- **Solution:** Solve the equation system on the coarsest level.

Each part of the multigrid algorithm is necessary to obtain an effective solution process. Fig. 6.3 clarifies the level and time of each execution step for a V-cycle and the Alg. 6.1 and 6.2 represent the multigrid algorithm for a V-cycle. Additional stopping criteria or passing of parameters as the number of smoothing steps are omitted for reasons of clarity.

**Figure 6.3:** Multigrid V-cycle from coarse to fine grid and back

The computational cost is considerable smaller for each coarser level. The number of grid points is reduced by about one half (1D), 3/4 (2D) and 7/8 (3D). The upper bound for the storage cost for a d-dimensional grid with n^d points, where $n = 2^l$, is ([Bri87])

$$n^d(1 + 2^{-d} + 2^{-2d} + 2^{-3d} + \dots + 2^{-nd}) < \frac{n^d}{1 - 2^{-d}} \quad (6.4)$$

which has to be multiplied by the memory needed for one level.

6.2.1 Smoothing

$$S(\mathbf{r}^l = \mathbf{K}^l \mathbf{e}^l) \quad (6.5)$$

Some iteration steps (n_{pre} and n_{post}) of an iterative solution method are performed to smooth (S) the residual (pre-processing) and the solution (post-processing). At least one pre-processing smoothing step is necessary. This step smooths the high frequent errors which cause slow convergence behavior in the multigrid algorithm. Good smoothing methods are direct iterative methods such as the weighted Jacobi method. The weighted Jacobi method is used with a heuristic weighting factor $\omega = \frac{2}{3}$.

The influence of the number of smoothing steps is considered in the examples in the corresponding sections.

6.2.2 Restriction and prolongation

The restriction (R) transfers the given and unknown vectors to the next coarser level

$$\mathbf{r}^{l+1} = R(\mathbf{r}^l). \quad (6.6)$$

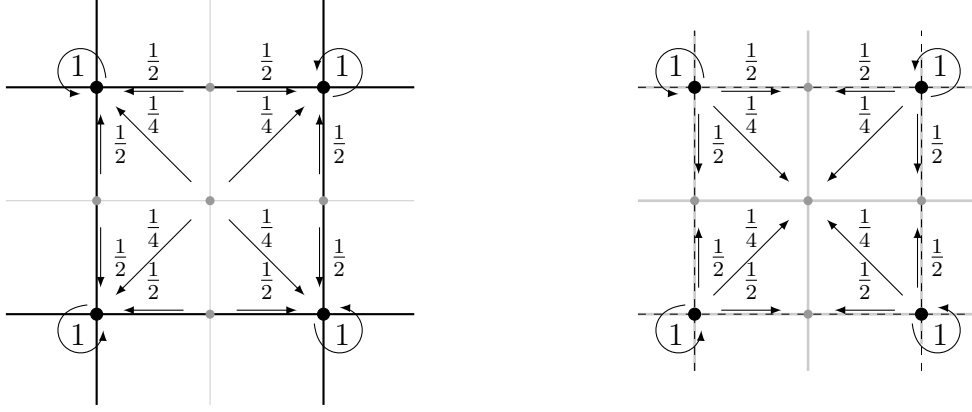


Figure 6.4: Restriction (left) and prolongation (right) in 2D from finer to coarser grid

Degrees of freedom which exist in both level are contained, intermediate dofs of the finer level are shifted to the neighbor dofs of the coarser level multiplied by a corresponding weighting factor.

The prolongation (P) maps the coarser level back to the following finer level and hence is the counterpart of the restriction:

$$\mathbf{v}^l = \mathbf{v}^l + P(\mathbf{v}^{l+1}). \quad (6.7)$$

There are different types of restriction and prolongation operators. The operator implemented and applied in this work is the full stencil which is the most complex and complete type. All reduced stencils such as one-directional (linear) or omitting the transition of the intermediate nodes loose some information from level to level. In the 2D case the full restriction is a 9-point stencil (Fig. 6.4)

$$\mathbf{R} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (6.8)$$

where as in 3D it is a similar constructed 27-point one. The prolongation operator is the transposed restriction operator except for the factor (2D: $P = 4R$).

6.2.3 Solution

On the coarsest grid the error equation

$$\mathbf{K}^n \mathbf{e}^n = \mathbf{r}^n \quad (6.9)$$

is approximated with an arbitrary direct or iterative solver. In this work the solver is the Jacobi preconditioned conjugate gradient method.

Algorithm 6.1 SolveMG(\mathbf{z}, \mathbf{r}_0)

```

1: if MG is preconditioner then
2:    $\mathbf{r} := \mathbf{r}_0$ 
3: else
4:    $\mathbf{r} := \mathbf{r}_0 := \mathbf{f} - \mathbf{K}\mathbf{u}$ 
5: end if
6:  $\mathbf{e} := 0$  and  $cycle = 0$ 
7:  $\epsilon_0 := \mathbf{r}_0^\top \mathbf{r}_0$ 
8: while  $cycle++ \leq maxCycles$  or  $\epsilon > tol^2 \epsilon_0$  do
9:   SolveMGLevel( $l, \mathbf{r}, \mathbf{e}$ )
10: end while
11:  $\mathbf{z} := \mathbf{e}$ 

```

Algorithm 6.2 SolveMGLevel($l, \mathbf{r}, \mathbf{e}$): solve one level

```

1: if  $l = \text{coarsest grid}$  then
2:   Solve  $\mathbf{r} = \mathbf{K}\mathbf{e}$ 
3: else
4:    $S(\mathbf{r} = \mathbf{K}\mathbf{e})$   $n_{pre}$  times
5:    $\mathbf{r}^{l+1} := R(\mathbf{r})$ 
6:    $\mathbf{e}^{l+1} := 0$ 
7:   SolveMGLevel( $l+1, \mathbf{r}^{l+1}, \mathbf{e}^{l+1}$ )
8:    $\mathbf{e} := \mathbf{e} + P(\mathbf{e}^{l+1})$ 
9:    $S(\mathbf{e})$   $n_{post}$  times
10: end if

```

The multigrid algorithm is described in the Alg. 6.1 with the function *SolveMGLevel*($l, \mathbf{r}, \mathbf{e}$) separately in Alg. 6.2.

6.3 Multigrid preconditioned conjugate gradient method

The multigrid method presented in the last section can be applied as a standalone solver as well as a preconditioner for an other iterative solver as the implemented conjugate gradient method. In the multigrid method the coarse grid correction smooths the low frequency errors over the whole domain and the smoothing with stationary iterative methods (e.g. Jacobi) damp the high frequency errors. Through the combination the multigrid method presents an efficient solution method as well as an efficient preconditioner for the CG: MGCG. The algorithm for the MGCG including matrix-free formulations is shown in Alg. 6.3.

Algorithm 6.3 EBE-MGCG

```

1: Choose a start vector  $\mathbf{u} \in \mathbb{R}^n$ 
2:  $\mathbf{r} := \mathbf{r}_0 := \mathbf{f} - \sum_e \mathbf{T}^\top E^e \mathbf{K}^0 \mathbf{T} \mathbf{u}$ 
3: SolveMG( $\mathbf{z}, \mathbf{r}$ )
4:  $\mathbf{d} := \mathbf{z}$ 
5:  $\alpha_1 := \mathbf{r}^\top \mathbf{z}$ 
6:  $\epsilon_0 := \mathbf{r}^\top \mathbf{r}$ 
7: while  $\epsilon > tol^2 \epsilon_0$  do
8:      $\mathbf{h} = \sum_e \mathbf{T}^\top E^e \mathbf{K}^0 \mathbf{T} \mathbf{d}$ 
9:      $\alpha := \frac{\mathbf{d}^\top \mathbf{h}}{\mathbf{d}^\top \mathbf{d}}$ 
10:     $\mathbf{u} := \mathbf{u} + \alpha \mathbf{d}$ 
11:     $\mathbf{r} := \mathbf{r} - \alpha \mathbf{h}$ 
12:    SolveMG( $\mathbf{z}, \mathbf{r}$ )
13:     $\beta_1 := \mathbf{r}^\top \mathbf{z}$ 
14:     $\beta := \frac{\beta_1}{\alpha_1}$ 
15:     $\mathbf{d} := \mathbf{z} + \beta \mathbf{d}$ 
16:     $\alpha_1 := \beta_1$ 
17:     $\epsilon := \mathbf{r}^\top \mathbf{r}$ 
18: end while
    
```

6.4 Examples: 3D-plate with circular hole and random material distribution

The first example is a three dimensional plate with circular hole which thickness is thus to get a cubic model (*plate* Fig. 6.5 left, dim=128). Secondly an artificially constructed cube filled with up to 75% random distributed material voxels and the remainder with void voxels (*random* Fig. 6.5 right, dim=32) is considered. The connectivity of the elements in this model is ensured through a ITK filter with a face-connectivity condition.

The examples are considered with different model resolution ranging from 32 to 256 voxels per dimension (dim). The dofs range from 110000 to 41 Mill. and are listed in Tab. 6.6.

The models are loaded with a given uni-axial displacement-driven traction in z-direction $u_z = 10^{-2} z_{max}$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = z_{max}$ and constrained by displacement boundary conditions: $u_x = 0$ for $x = 0, y =$

dim	dofs	
	random	plate
32	107730	90288
64	823596	677820
96	2737422	2240700
128	6439197	5239980
192	-	17490432
256	-	41180652

Figure 6.6: Degrees of freedom for the random distributed material (*random*) and the 3D-plate with circular hole (*plate*) for different model discretizations, given in voxels per dimension (dim)

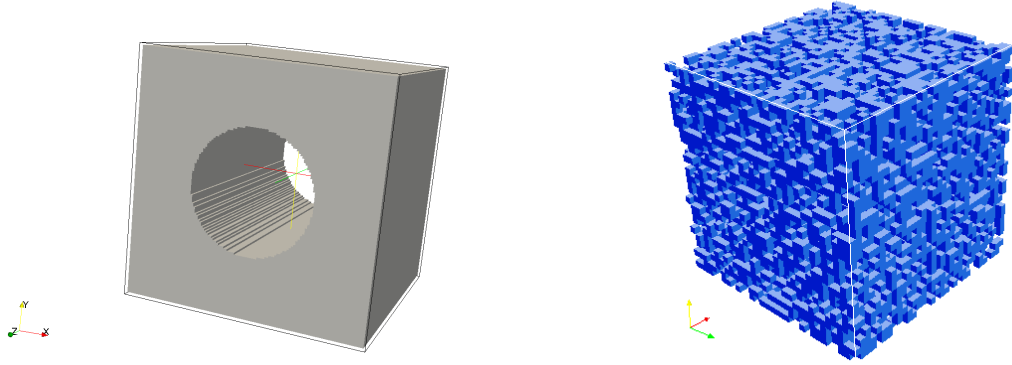


Figure 6.5: Three dimensional plate with circular hole with 128 voxels per dimension (left) and cube with 75% random material voxel-based elements with 32 voxels per dimension (right)

$\{0; y_{max}\}, z = \{0; z_{max}\}; u_y = 0$ for $x = \{0; y_{max}\}, y = 0, z = \{0; z_{max}\}$ and $u_z = 0$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = 0$. $\{\cdot\}_{max}$ is the maximal value of the model dimension in $\{\cdot\}$ -direction. The dimension of one voxel is 1 mm in each direction. The linear elastic material parameter are $E = 100000N/mm^2$ and $\nu = 0,2$.

6.5 Benchmark

The characteristics of the proposed multigrid method in the matrix-free voxel-based analysis is evaluated in this section. The results are presented relative to the analysis with the Jacobi preconditioned conjugate gradient method (JCG) of the respective FE model with the highest resolution. The present benchmark is based on the row-major data structure and was computed on a workstation with a 2nd generation Intel core I7-2600 processor (8 M Cache, 2.6 GHz) and 8 GB (2x4 GB) RAM. The MG algorithm requires the setting of several variables which influence the problem specific convergence behavior. These are the number of pre- and post smoothing steps, the number of V-cycles and the number of grid levels. For the presented EBE-MGCG the multigrid parameter which leads to the fastest computation time for a wide range of microstructural geometries were evaluated in the following benchmarks. The chosen multigrid cycle is a V-cycle which is despite it's simplicity the fastest cycle for the MGCG [Häf07].

6.5.1 Memory demand

The computational cost for each coarser level is considerable smaller with the highest reduction in 3D. The number of grid points is reduced by about 7/8 per level. The upper bound for the storage cost [Bri87] $8/7n^3$ is multiplied with the data stored at

one node. For the row-major voxel grid with n^3 nodes eight 64-bit precision vectors are allocated for: the displacements ($3n^3$), the residual ($3n^3$), the diagonal preconditioned vector ($3n^3$), Young's modulus (n^3), three 32-bit precision vectors for the identifying of the nodes and elements, one 1-bit vector for boundary conditions ($3n^3$) and the basis element stiffness matrix (24×24) and some additional values. This leads to 125 bytes per node. The example *plate* with $\text{dim}=256$ has about 13.73 Mill. nodes and therewith the memory demand yields to 2.6 GB for the finest grid and to 3 GB for the multigrid model. In the computation of this grid model with JCG 4 GB main memory and with MGCG 5.9 GB main memory were used which includes loaded shared libraries and the memory demand of the solution algorithm. For the ratio of the MGCG to CG $5.9/4 = 1.475$ the theoretical bound can not be reached. Nevertheless the memory demand stays considerable lower compared to the corresponding PCG analysis with the commercial software ANSYS which uses 101.2 GB and is not feasible with the average desktop computer (Tab. 6.1). The solution time and pre-processing time of the ANSYS analysis are considerable higher as well, though for the pre-processing it is not presented here. The analysis using and comparing to ANSYS was performed on a Intel Xeon with 20 MB Cache, 2.20 GHz and 512 GB RAM, which is required due to the large memory demand of the analysis in ANSYS. Further, the termination criterion for these analysis is set to $\|\mathbf{r}\| = 10^{-8}$ following the standard ANSYS settings whereas the other benchmarks were conducted with $\|\mathbf{r}\| = 10^{-6}$ as termination criterion.

<i>plate</i> (dim=256)	solution time (min)	memory (GB)
MGCG6111	18.4 (100,0%)	7.6 (100%)
ANSYS	115.9 (629.9%)	101.2 (1332%)

Table 6.1: Computation time and memory demand of the *plate* (dim=256) analysis with 41.1 Mill. dofs performed with the PCG solver of the commercial software ANSYS and with the proposed MGCG solver with 6 grid levels and one cycle and pre- and post-smoothing step

6.5.2 MG vs. MGCG

The 3D-plate with circular hole (*plate* Fig. 6.5 left) with a discretization of 256 voxels per dimension and about 41 Mill. dofs is solved with the JCG as reference solution method, the MG as standalone solver and finally with the MG as preconditioner (MGCG). In Fig. 6.7 the computation time of the analysis of this model is presented for different solution methods relative to the JCG. The MGCG is slightly faster than the MG standalone solver even for unfortunate multigrid parameter settings. Furthermore the MGCG is more robust for heterogeneous models with high ratios in the material parameter [Häf07].

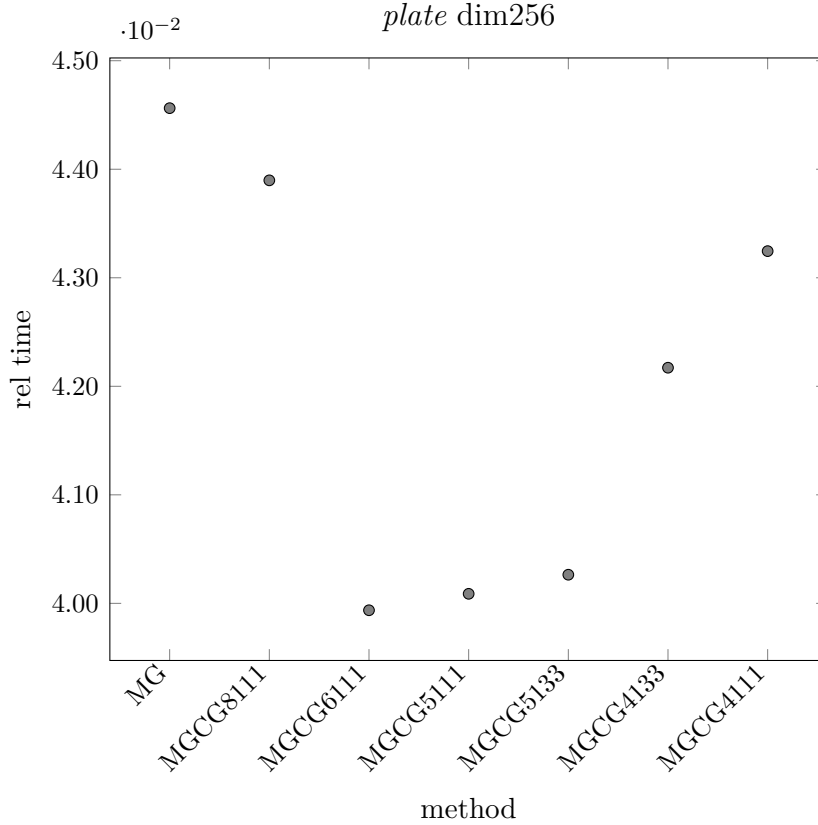


Figure 6.7: Computation time of MG and MGCG relative to JCG method for the *plate* example with 256 voxels per direction

In the MG analysis with 8 multigrid levels which correspond to a maximal coarsening with 2 elements per dimension, the MG algorithm needs only 8 V-cycles to converge (with 3 pre-and post-smoothing steps) and applying the same coarsening as preconditioner (with one pre-and post-smoothing step) the MGCG algorithm converges to a standalone MG algorithm with similar solution times.

6.5.3 Number of multigrid cycles

The influence of the number of multigrid cycles on the computation time is evaluated in Fig. 6.8 for the cube with randomly distributed material (*random*) and in Fig. 6.9 for the 3D-plate with circular hole (*plate*). The results for four model resolutions and for a varying number of smoothing steps are presented. The multigrid method is applied with the maximal number of hierarchical level for each model discretization respectively.

For the described configuration the multigrid preconditioner is the most effective if one multigrid cycle and only one smoothing step is applied for all discretizations of the *plate* and for the smaller discretizations of the *random* examples. However for the larger *random* discretization the configuration with 3 smoothing steps and one cycle is faster. This is explained by the heterogeneous microstructure which needs more smoothing to

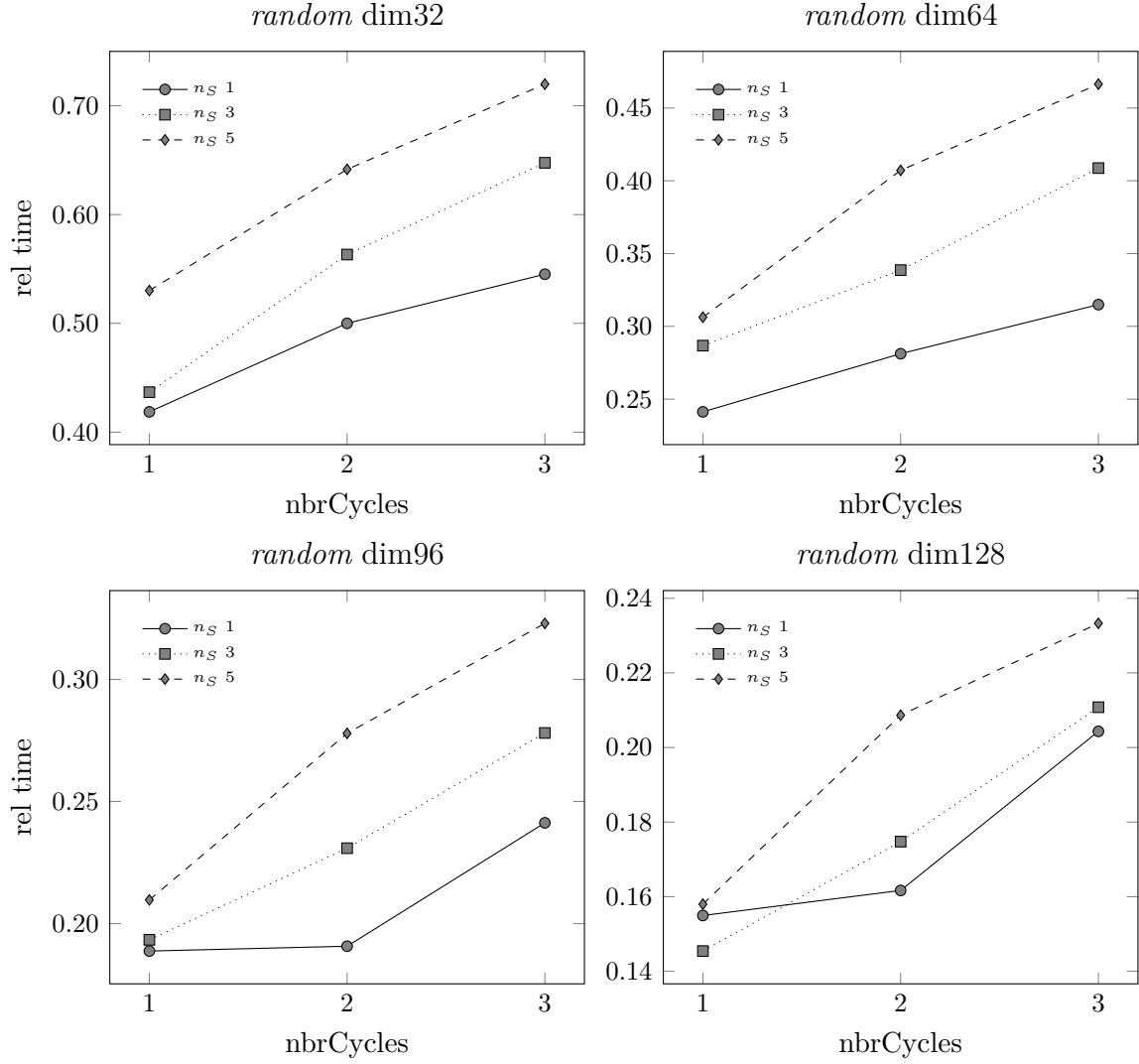


Figure 6.8: Computation time vs. number of multigrid cycles (nbrCycles) relative to JCG analysis for *random* example with 4 discretization, n_S is the number of applied smoothing steps

reach similar damping ranges than the homogeneous material. Nevertheless the computational cost for the smoothing step remunerate only for the bigger model.

The influence of the number of smoothing step is further investigated on the *plate* example for a multigrid preconditioner restricted to 3 grid levels. Fig. 6.10 shows the results. The fastest analysis is reached with one cycle and three smoothing steps though the variant with five smoothing steps improves with the model size. Consequently the multigrid preconditioned method is recommended with one cycle. For the number of smoothing step the model size, the number of multigrid level and the heterogeneity of the problem have to be considered. The smaller the number of multigrid levels and the higher the model heterogeneity is, the higher the number of smoothing steps has to be. Generally for smaller than three multigrid cycles and a model size larger than 5 Mill. dofs

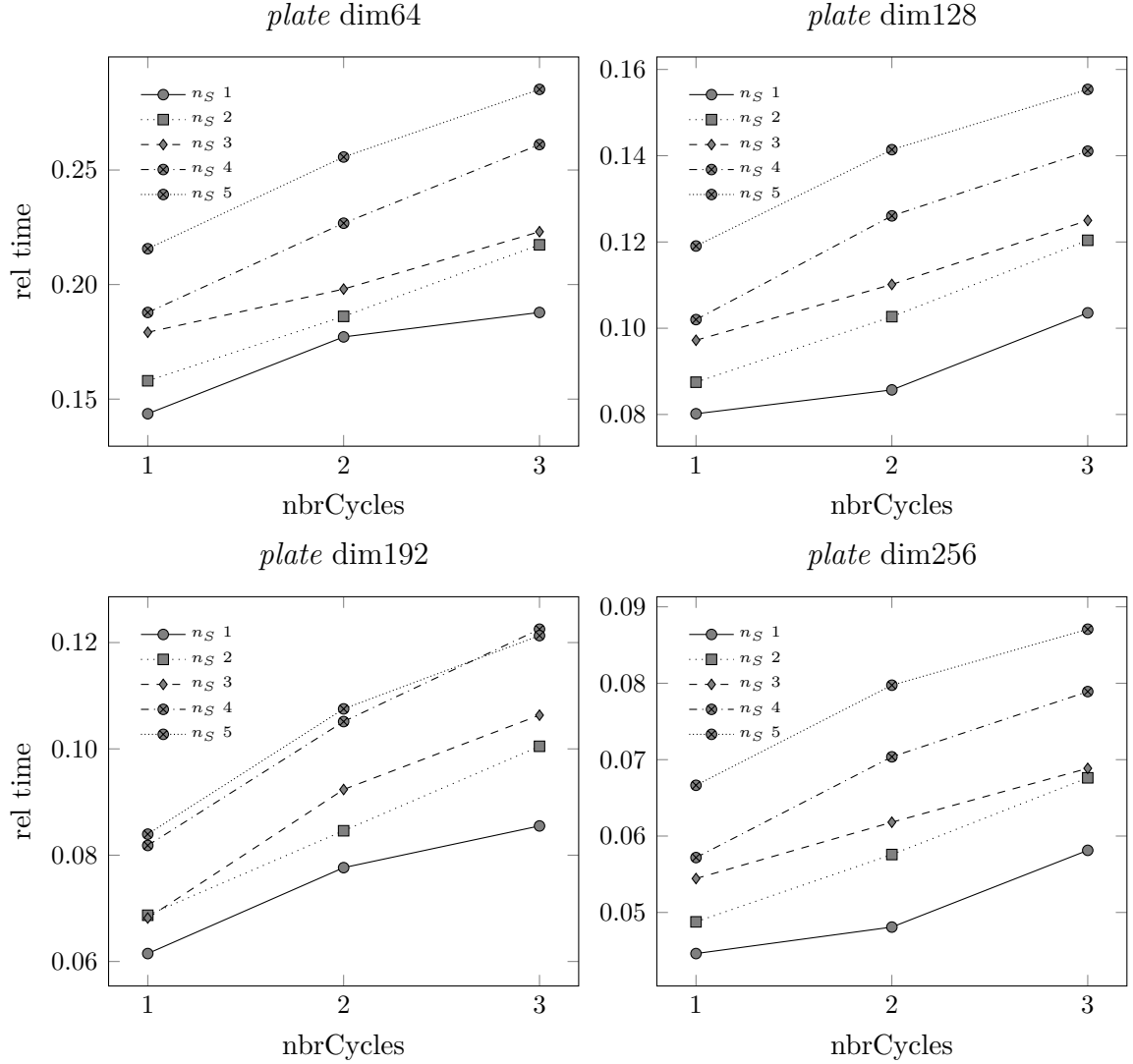


Figure 6.9: Computation time vs. number of multigrid cycles (nbrCycles) relative to JCG analysis for *plate* example with 4 discretization, n_S is the number of applied smoothing steps

three smoothing steps are recommended, for smaller models and more multigrid levels one smoothing step. Furthermore the computation time decreases relative to the Jacobi preconditioned solution with the increasing model size which approves the relevance of the MGCG for matrix-free voxel-based finite element models.

6.5.4 Number of multigrid levels

For the *plate* example with a discretization of $\text{dim}=192$ the number of multigrid levels used in the multigrid analysis is varied. The multigrid algorithm requires at least 3 grid levels to deploy its effectiveness (Fig. 6.11), whereas the differences in computation time according to varied cycle or smoothing steps numbers is small. For visibility reasons the

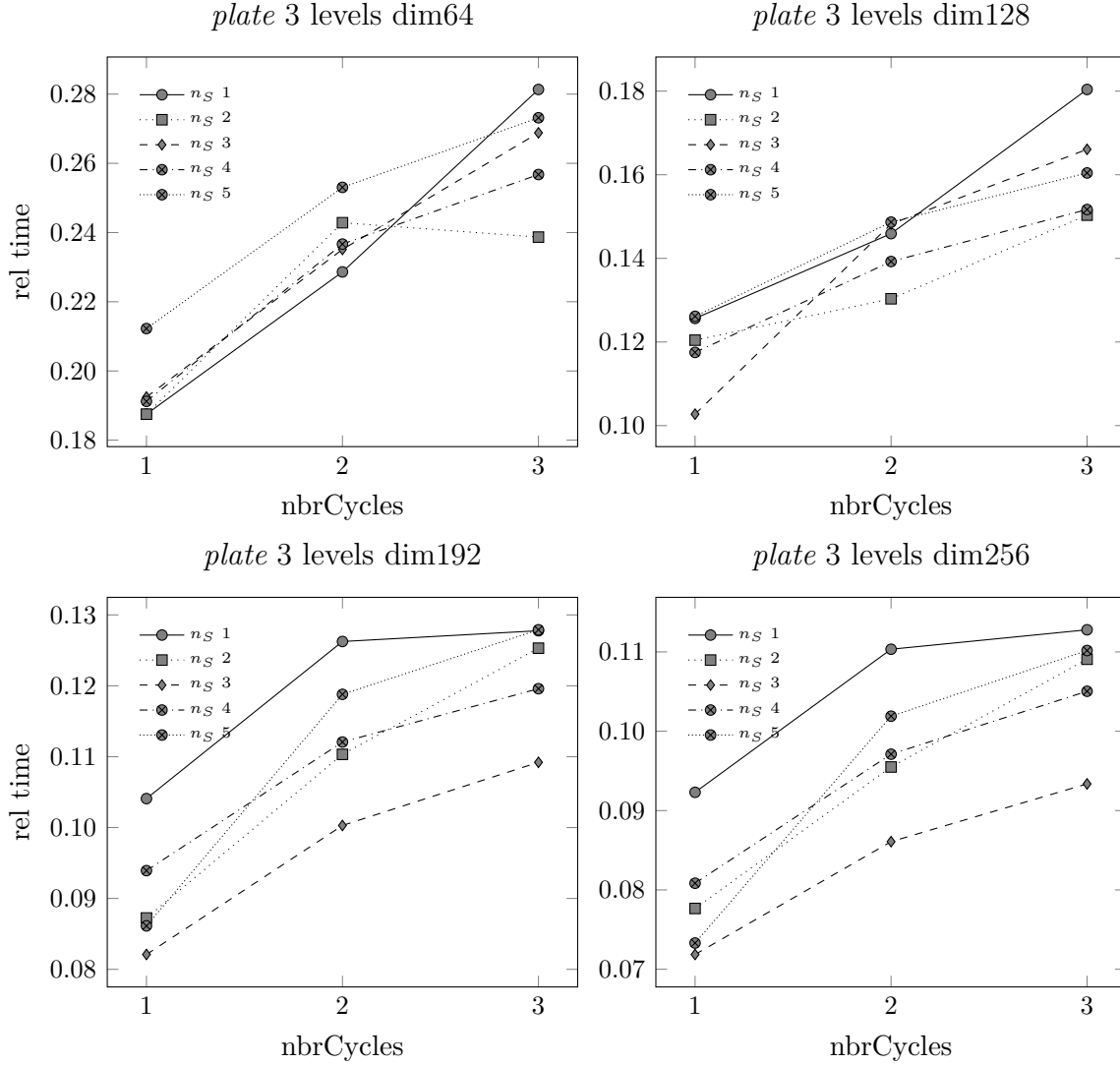


Figure 6.10: Computation time vs. number of multigrid cycles (nbrCycles) relative to JCG analysis for *plate* example with 4 discretization, number of multigrid levels is restricted to 3, n_S is the number of applied smoothing steps

Fig. 6.11 is reduced to the computational effective number of multigrid levels starting with three levels and to the faster analysis with one multigrid cycle but with 1 to 5 smoothing steps in Fig. 6.12. The MGCG leads to the best results with 4 and 5 multigrid level for this example. The smallest level (7) has only 9 elements which is too small to compensate the effort of the grid transfer operators. The level 5 includes 6591 dofs, for coarser levels the transfer effort exceeds the effort for the solution. The multigrid level should not be smaller than 10 voxels per dimension. Generally such small grids are prevented through the limited divisibility through 2 required for the coarser grid level generation. The described inverse effect of the number of levels versus the number of smoothing steps is also visible in this figure. The examples with higher number of smoothing steps generally gain in terms of the computation time with the decrease of the number of levels.

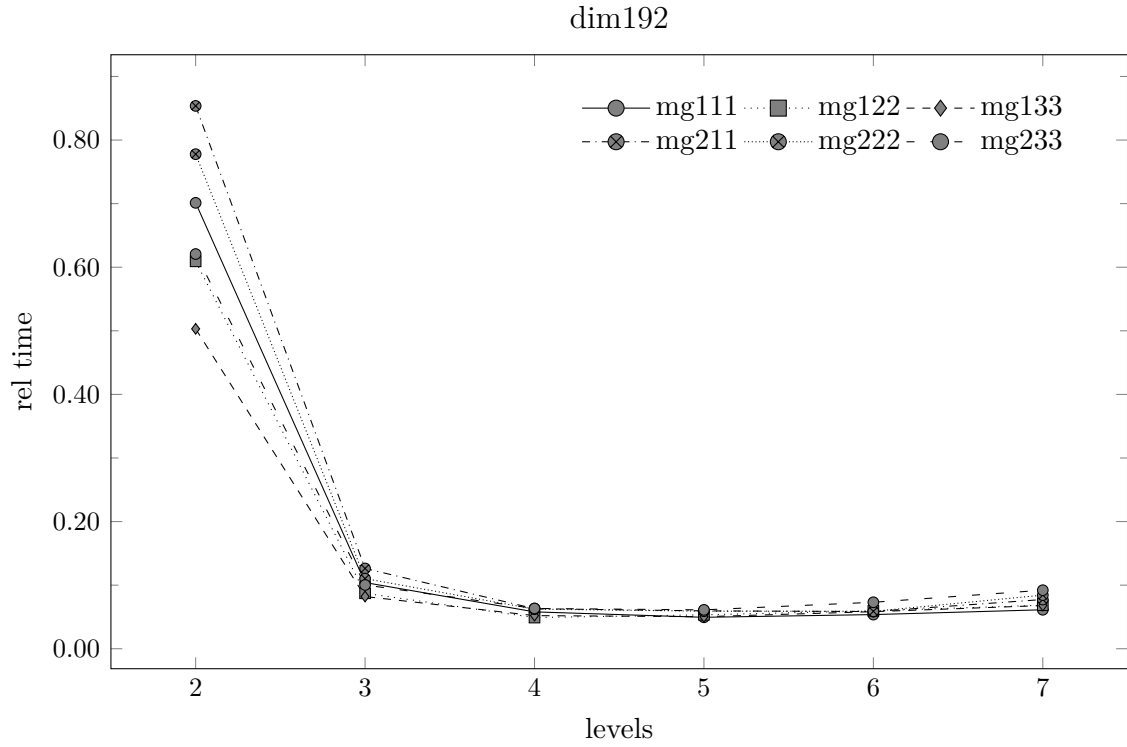


Figure 6.11: Computation time vs. grid levels relative to JCG analysis for *plate* example with discretization of 192 voxels per direction for grid levels from 2 (min) to 7 (max) grid levels for 1 to 2 V-cycle and 1 to 3 numbers of smoothing steps

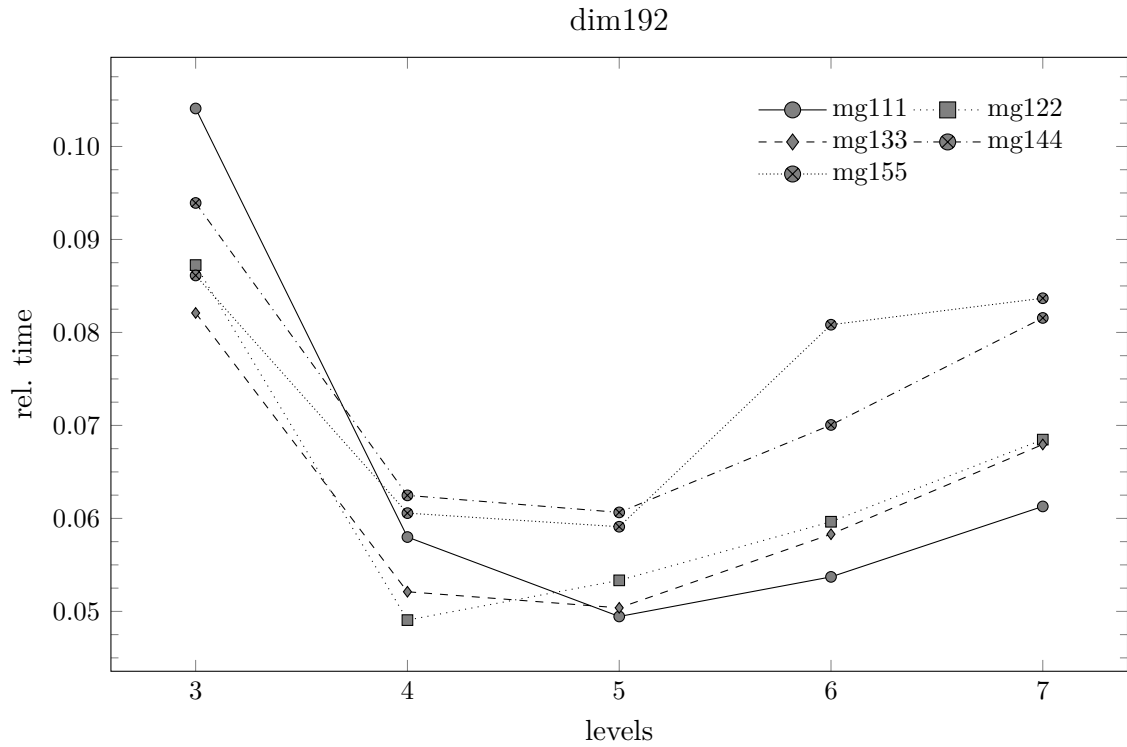


Figure 6.12: Computation time vs. grid levels relative to JCG analysis for *plate* example with discretization of 192 voxels per direction for grid levels from 3 to 7 grid levels for 1 V-cycle and 1 to 5 numbers of smoothing steps

Chapter 7

Mesh adaption based on octree algorithm

Image based voxel data sets with fine resolutions as considered in this work get in the range of 100 to 1000 millions dofs. One method to reduce the amount of data is to apply octree algorithms to coarse homogeneous material regions. Octree algorithms are part of hierarchical data structures which aim in sorting data efficiently and focus on interesting data subsets [Sam84]. The implemented octree is only structured through the Morton key, no additional tree information is stored. This type of octree is called pointer-less [FA11a]. The Morton order is the best suited code to represent spatial hashed octrees where the pointers are replaced by index representations [Lew+10] and is described in Sec. 2.3. This chapter gives a short overview over octrees (3D)¹ focused on the geometrical and mechanical applications. The implemented octree algorithm is described and the special features are highlighted.

7.1 State of the art

Traditionally hierarchical data structures are used for geographic information systems, image processing, computer graphics, computational geometry and robotics. The basis is the principle of recursive decomposition [Sam84]. The data is sorted in a tree-like structure: Each point is a node of the tree. The top node is called root and is only a parent node. A parent node has children which can also be parents. A node without children is called leaf. For general information on hierarchical data structures it is referred to the textbook [MS05] and specific for spatial data structures to [Sam94; Sam06; NW00].

Finite element applications have to be sorted with respect to the location of the data which leads to the spatial data structures. The best suitable for grid-based data are quadrees for two-dimensional spatial data where each parent has four children or octrees for three-dimensional spatial data with 8 children per parent.

¹Quadrees are the equivalent of octrees in 2D. They are sometimes used for visualization purpose.

Furthermore the data structures differ through the type of information access: pointer-based or lookup tables. Pointer-based data only allows access to the child through the pointer at the parent (parent-child-pointer) or similarly through siblings for siblings-pointer [Sam84]. With lookup tables the data is directly accessible through a unique key for each data point and stored in a linear field and for this reason is named linear quadtrees or linear octrees. Pointer-less octrees and quadtrees are described in [Gar82b] and [Gar82a] respectively for black-white images with quaternary code describing the location with respect to the four corners (2D). A Hilbert space filling curve as lookup table key is used in [GZ99] and the Peano curve in [MWZ06] for adaptive multigrid methods. Though the data sorting through the Morton keys is the most pertinent choice for quadtrees/octrees on grid-data. The Morton key can be computed directly from the location of the element or nodes through interleaving bits [Lew+10] and vice versa the Morton key is directly linked to the location of the data. The hierarchical structure is then saved in a one-dimensional array by traversing the tree depth-first. The data locality is provided naturally through sorting. This aspect of the octree data structure is applied for multigrid based iterative solutions and uniform voxel models in parallel [FA11b]. The Morton order is described in detail in Sec. 2.3.

The main advantage of pointer-less representation vs. pointer-based is the significantly less data amount and direct accessibility. E.g. for image processing in [Cho+09] it is shown that the memory demand of the used link-less quadtree is only one forth of other pointer-based trees, this comes with the disadvantage of the lack of adaptivity in the lookup tables considering changing data and slightly disadvantage at the velocity of random data access.

Not every spatial data structure is suitable for finite element applications. From the numerical calculation point of view it is clear that neighboring elements which differ considerably in size leads to numerical problems due to hanging nodes even if from the hierarchical data structure point of view the amount of data is most efficiently saved. Hanging nodes are mesh irregularities which are not suited for finite element schemes and are solved by additional constraint equations. For finite element applications of grid models the quadtrees/octrees are mostly implemented with a 2:1 restriction [SB10]. Since in this work the cubic element shape has to be conserved for the matrix-free method, octree adaptivity methods which avoid hanging nodes [ISS08] can not be applied. The Mortar method with Lagrange multipliers is used for hanging nodes [Bel99], but is not suitable for matrix-free methods. To solve the hanging node problem in [SB10] an adapted ansatz-functions depending on the location of the hanging node in the element is introduced. For stress analysis of image data quadtree/octree refinement discretizations are applied to the scaled boundary finite element method to omit hanging nodes naturally [Sap+17]. In [Zan+15] a hierarchical, multi-level hp-formulation is introduced which uses the higher order ansatz function to introduce compatibility for the octree construction.

In FE calculations the information of the nodes of one element is mandatory classical provided by an element-to-node-table. For the grid-based approach with the standard format of row-based data fields the information of the neighbor nodes can also be directly calculated through relative neighbor locations which are equal for every element. This is not provided by the octree, the neighbors have to be detected through a neighbor search. The pointer-less octrees get this information through a binary search in the lookup table. This corresponds to a search from the root down to the leaves. In [Cas+08] a statistical approach is presented which uses a special octree construction from the root node as middle node and the appending of keys in all directions for the children. The search is then adapted to a faster statistical approach which starts from the same level, ascends the tree and then descends to the searched node. The automatic detection of child nodes can not be adapted for this work where the octree node corresponds to the first finite element node of the element. Though the octree in [Cas+08] is based on the Morton code with interleaving bits, the bit information of the level is added to the key to enable the proposed optimized searches. This leads to huge key numbers for the finest grid and the data of coarser elements is stored before all the finer levels. So the lookup table loses the ordering according to the location in space which is important for cache-efficient algorithms and for domain decomposition techniques. Nevertheless [FA11b] adapt the ascend-descend approach to access the neighbor nodes to an exponential interval search combined with a binary search for applications of multigrid procedures without coarsened elements in original mesh.

7.2 Adaptive mesh construction

The octree data structure implemented in this work conserves the properties of the Cartesian mesh (voxel-based elements), the matrix-free computation type and provides data locality for optimized cache efficiency. The construction of multigrid levels for multigrid algorithm is then straightforward. Different pre-requirements have to be followed:

- **Fine resolution boundaries** All material interfaces and all model boundaries maintain the fine resolution, so the material boundaries are described in detail and different boundary conditions can be applied independently of the mesh. This is important for more complex boundary condition such as point loads or non-uniform surface loads. It conserves the original loading conditions.
- **Morton key coarsening** The coarser element is always a parent node of the octree, therewith for all coarser elements the Morton key is divisible through 8^l where l is the corresponding depth of the octree node.
- **2:1 restriction** The coarsening is restricted to a 2:1 octree. The difference between neighbor elements can be at most one level. This implies that on one element edge

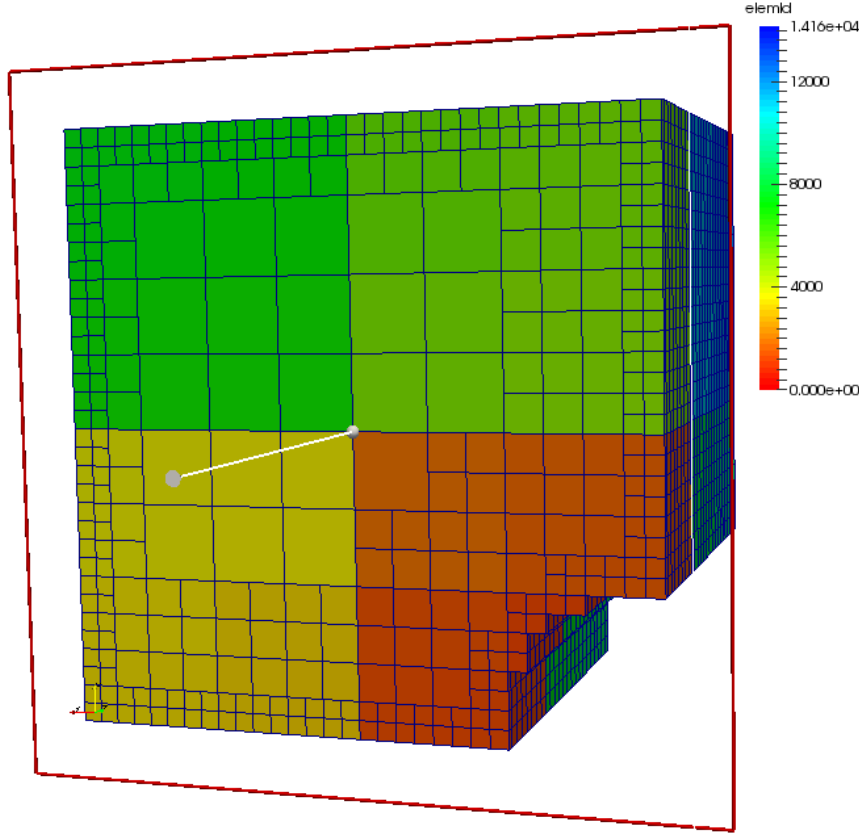


Figure 7.1: Resulting octree grid of a cube with void quarter tube with resolution $dim = 32$ and with colored element identifier

or plane only one hanging node can exist and is handled by constraint conditions. Therewith the finite element solution is kept numerical robust.

The Morton key identifies the octree *node*¹. Each octree *node* consists of the id of the *node* and of a weight w and two integers. For solely finite element nodes the weight is zero ($w = 0$). If the weight is bigger zero ($w > 0$) the octree node corresponds to the finite element and the first finite node of this element with the local index 0. Then there is an integer saving the different kind of constraints of this finite node which differs bit-wise (Tab. 7.1) and another for the smallest number of the grid level where the node exists (e.g. 0 for a node which exists on the finest grid).

The construction of the octree data structure starts with the finest grid ($l = 0$) from the data of the VTK image. Contrary to the literature where the coarsest grid level has generally the number 0, here it is the finest level as in Ch 6. The algorithm of the octree construction (Alg. 7.1), checks if each element with a key divisible by 8^{l+1} has seven next neighbors with the same material and all criteria for 2:1 octree are met. If this is true the neighbors are deleted and the checked element is coarsened to the next grid level.

¹The vocabulary *node* is here used in the context of the octree, not in the finite element sense.

The next seven neighbors are identified through the Morton key which represents also the nodes of the *key* element checked for coarsening. The resulting octree is presented in Fig. 7.1, the element identifier is represented through the displayed color range.

The matrix-free algorithm can be applied to the coarser octree elements in the same way as for the fine grid elements. The resulting octree is saved pointer-less in a lookup table through the Morton key presented in Ch. 2.3. For the matrix-vector product on element level the node information has to be computed by binary search contrary to row based models where the information can be accessed through simple and equal neighbor relations. The binary search is combined with the in [FA11b] proposed adapted search shortening the search region in the lookup table to the region of the one level coarser z-curve of the Morton code the starting node lays within. For the hanging node correction additional neighbors in negative coordinate direction have to be considered so the adapted search region of the lookup table is further widened to the two levels coarser z-curve. The number of nodes belonging to one parent node (one z-curve) is 8^{2^l} .

Algorithm 7.1 One level of octree construction

```

1: for each key in level l do
2:     if  $key \% 8^l == 0$  and  $weight.key > 0$  and  $key < n_{key}$  then
3:         Check, if weight of next 7 neighbor elements is equal
4:         if true then
5:             Check, if level of neighbor elements suffices 2:1 grid requirement
              and if weight is equal
6:             if true then
7:                 Create new coarse grid element and delete 7 fine grid
                  neighbor element of key element
8:             end if
9:         end if
10:    end if
11: end for

```

7.2.1 Saving of displacement constraints and hanging nodes on edges or planes in one integer

Each octree node has stored one integer for displacement constraints and for the direction of the plane or edge of the neighbor nodes for hanging node (HN) correction together. Octree nodes without special consideration have a zero value in the variable. In the purpose of memory demand the data is stored bit-wise¹ as described in Tab. 7.1.

To save the hanging node and the displacement constraint in one integer different bit arrays are used and the integer has to be at least six bits big, so here and also for the level number a 8 bit integer such as `uint8_t` is sufficient. The last 3 bits contain

¹considering big endian sorting

decimal	binary	type of constraint
0	000	not constraint
1	001	hanging node: x-direction-edge
2	010	hanging node: y-direction-edge
4	100	hanging node: z-direction-edge
3	011	hanging node: xy-plane
5	101	hanging node: xz-plane
6	110	hanging node: yz-plane
8	001 000	displacement constraint in x-direction
16	010 000	displacement constraint in y-direction
32	100 000	displacement constraint in z-direction
9	001 001	displacement constraint in x and HN in x-edge
29	011 101	displacement constraint in x and y and HN in xz-plane

Table 7.1: Displacement constraints integer: Bit-wise storing of constraint information and hanging node characteristic of one node

the displacement constraints and the 6th to the 4th bit from the back the hanging node information respectively. Therewith combination of both information is possible.

7.2.2 Solution algorithm considering hanging nodes

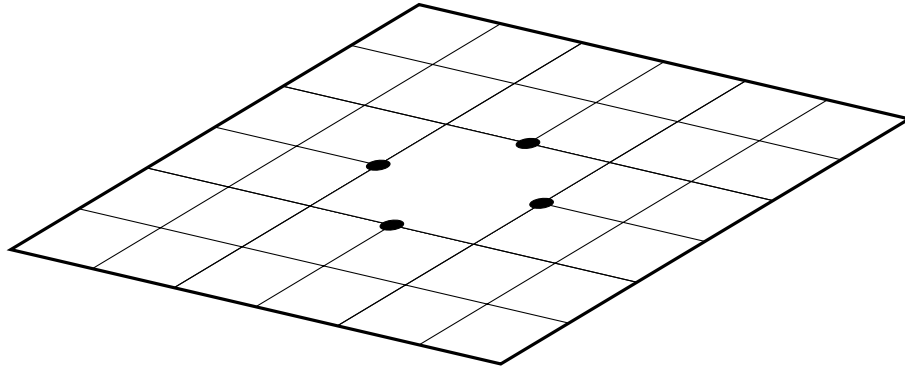


Figure 7.2: Quadtree: 2D grid with coarser middle element and associated hanging nodes (filled circles)

The introduction of the octree discretization leads to hanging nodes (Fig. 7.2) which introduce a irregularity to the finite element mesh and have to be constrained. This is done by a constraint displacement condition, the hanging nodes are computed considering the linear ansatz-functions of the coarse grid elements which is for this regular structure

the mean of the dofs of the coarse grid nodes on which edge

$$u_x^{HN} = \frac{u_x^{cI} + u_x^{cII}}{2} \quad (7.1)$$

or plane

$$u_x^{HN} = \frac{u_x^{cI} + u_x^{cII} + u_x^{cIII} + u_x^{cIV}}{4} \quad (7.2)$$

the hanging node is located. This is done in an independent step of the iterative process which adds computational effort compared to the homogeneous grid solution.

7.3 Example: Symmetric 3D-plate with circular hole

The three dimensional plate with circular hole and a symmetric part of this model are considered in the following benchmarks with different discretizations (Tab. 7.3) and solved with the JCG. The models are loaded with a given uni-axial displacement-driven traction in z-direction $u_z = 10^{-2} z_{max}$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = z_{max}$ and constrained by displacement boundary conditions: $u_x = 0$ for $x = 0, y = \{0; y_{max}\}, z = \{0; z_{max}\}$; $u_y = 0$ for $x = \{0; y_{max}\}, y = 0, z = \{0; z_{max}\}$ and $u_z = 0$ for $x = \{0; x_{max}\}, y = \{0; y_{max}\}, z = 0$. $\{\cdot\}_{max}$ is the maximal value of the model dimension in $\{\cdot\}$ -direction. The dimension of one voxel is $1mm$ in each direction. The linear elastic material parameter are $E = 100000N/mm^2$ and $\nu = 0,2$.

<i>sym</i>	
dim	dofs
32	102267
64	782145

Figure 7.3: Degrees of freedom for the symmetric 3D-plate with circular hole (*sym*) for different model discretizations (dim)

The analysis were performed on a Intel Xeon (20 M Cache, 2.20 GHz). In Fig. 7.4 the displacement solution for a model with a resolution of 32 voxel per dimension is pictured.

7.4 Benchmark: Space filling curve

In Sec. 2.3 the row-major order and the Morton order were introduced. The precedent chapters were all implemented based on row-major order. For memory-efficient thus pointer-less octree presentation *octree nodes* are saved in a lookup table and accessed through the Morton key. With the Morton key octree nodes of one parent are naturally neighbor nodes. Local data access is derived automatically. Nevertheless the neighbors have to be accessed through binary search where as in the row-major order the neighbors can be accessed directly through a constant neighbor relation. The same

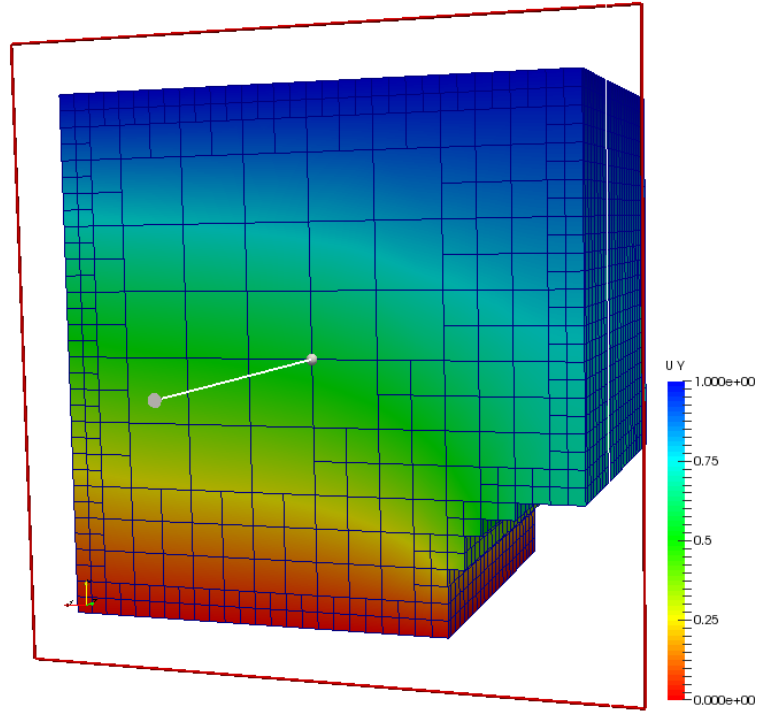


Figure 7.4: Displacement solution for an octree construction with determined pre-requirements for $dim = 32$ model

geometrical model sorted with row-major order and solved with JCG as reference example is compared to the Morton order without octree coarsening (one octree level and homogeneous discretization). The three dimensional plate with circular hole example from Sec. 6.4 is studied with different discretizations. The Fig 7.5 shows the relative computation time and relative memory demand vs. dofs of the Morton ordering based JCG analysis. The memory demand increases from about 30% up to about 45% from the coarse to the fine discretization respectively. The data structure is only changed from solely vector based element and node information to a lookup table based on the Morton key. Nevertheless the memory consumption is higher although generally it remains low for such a big problem. The time consumption increase about 55% up to about 70% from the coarse to the fine discretization respectively. This is considerably higher and is due to the additional computation of the nodes of the element through a binary-search. The adapted search algorithm is implemented with additional clauses to omit *undefined behavior* for the used lookup table and therewith could not improve the computation time of the neighbor searches.

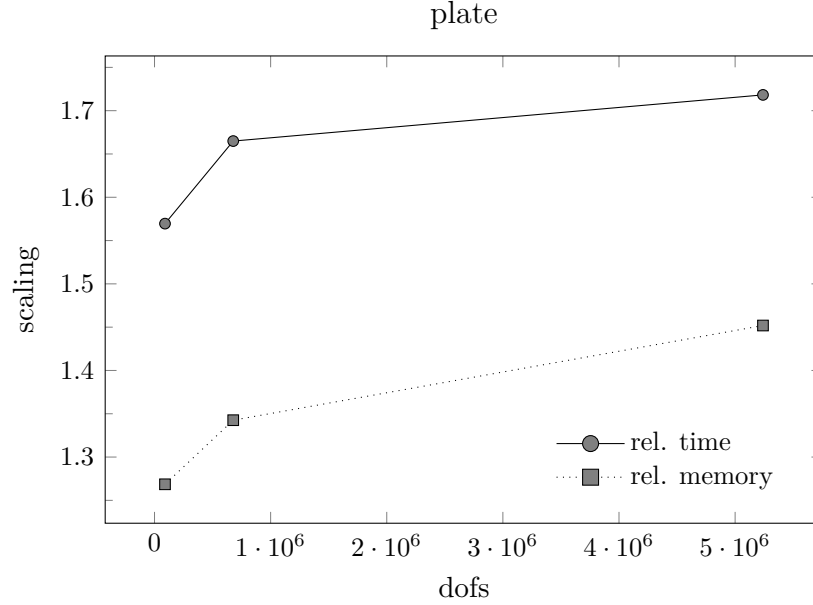


Figure 7.5: Computation time and memory demand vs. dofs for the Morton sorting relative to row-major sorting solved with JCG, the example is the *plate* example with different discretizations

7.5 Benchmark: Octree coarsening

The homogeneous material regions can be coarsened without losing information. For the symmetrical three dimensional plate with circular hole model (*sym*) with $\text{dim}=64$ the applied octree algorithm leads to a reduction of 70% of dofs compared to the uniform discretization whereas in the smaller model ($\text{dim}=32$) the reduction is about 50% due to the smaller homogeneous region (Fig. 7.6). The reduction in memory demand is slightly less (about 60% for $\text{dim}=64$), yet the reduction is considerable. However the decrease in model size and memory demand is dependent of the degree of material heterogeneity. Furthermore the time consumption due to neighbor searches increases considerably. The octree discretization requires the additional computation of the hanging nodes through the constraint conditions and therewith additional neighbor searches. The increase in computation time (Fig. 7.7 left) far exceeds the decrease in memory demand (Fig. 7.7 right). The slow-down is supposedly linked to the additional clauses in the search algorithm which are required to prevent *undefined behavior*. So far the implemented octree coarsening is not successful applicable.

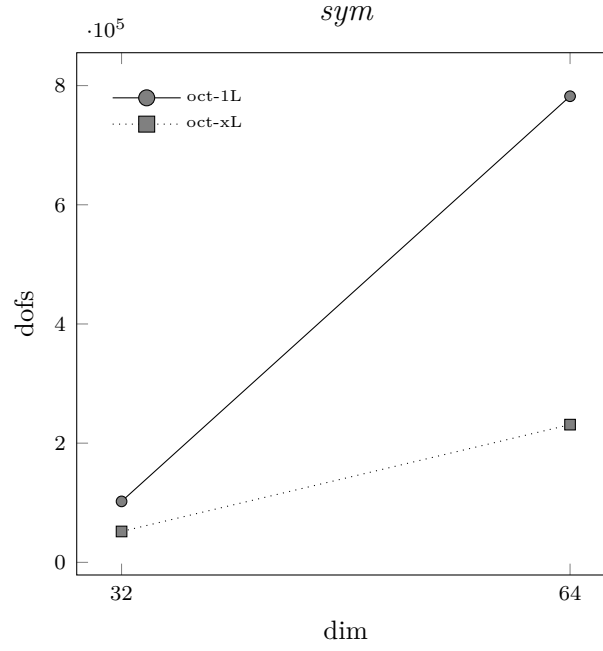


Figure 7.6: Dofs for the *sym* example with dim=32 and dim=64 with (oct-xL) and without (oct-1L) octree data structure

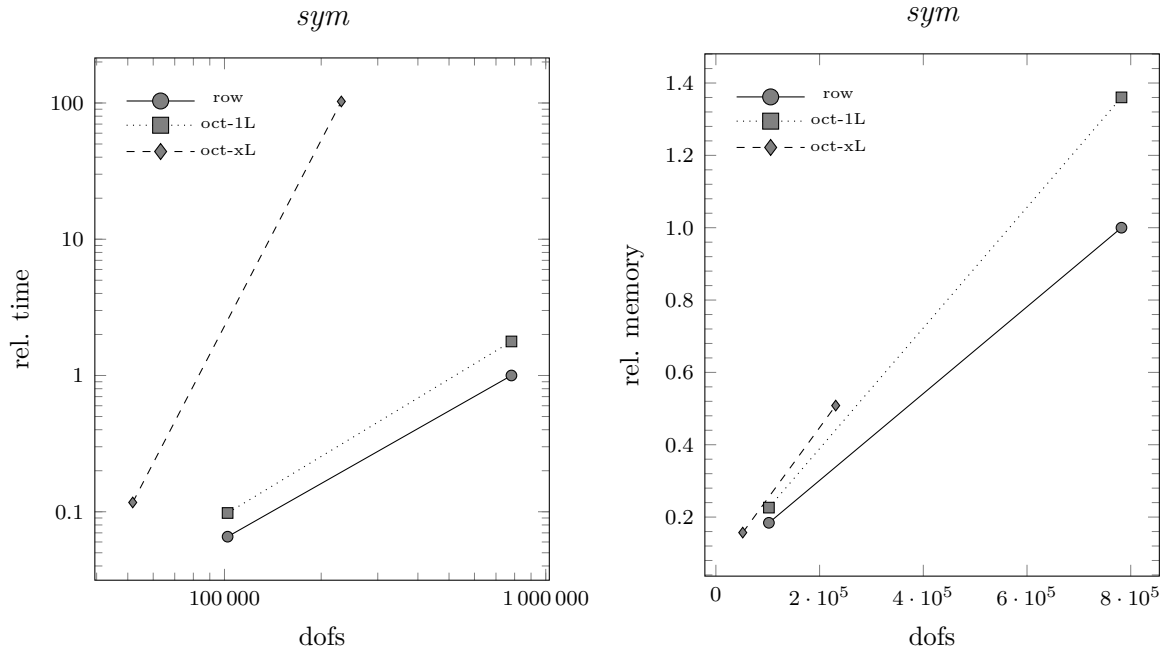


Figure 7.7: Rel. computation time and rel. memory demand for the row-major (row), Morton (o1L) and octree (oxL) based data structures, with the JCG analysis for the *sym* example with 32 and 64 voxels per direction relative to the larger row-based discretization

Chapter 8

Embedded boundary elements

One main aspect of this work is the matrix-free solution based on Cartesian grid elements. The Cartesian grid discretization leads to jagged material boundaries and thus to numerical introduced local stress peaks. To maintain the advantages and overcome this main disadvantage embedded boundary elements have been implemented in this work. Embedded boundary elements are finite elements which can map different materials in one element. There are various possibilities to model material interfaces and model geometries with indirect or implicit methods however the most are not compatible to the methods presented in this work. The chosen embedded boundary elements model material boundaries through variable material data at the integration points.

8.1 State of the art

From the mechanical point of view it is challenging to deal with the numerical discontinuities in the stresses due to the jagged boundaries. Though the voxel-based FE model accurately predicts the homogenized elastic material properties [Rie+95; Che+07] and the calculated Mises effective stress and strain distributions correspond well to experimental micro-damage regions [NCG05] and to experimental strain measurements [Grö+09], jagged boundaries produce an error to the model which is less than 10% for a sufficient fine discretization, e.g. for a trabecular bone model four voxel per trabecular [RWR93]. This criteria for the model resolution is also valid for damage modeling and leads to the same error in the solution [Mis05].

However, there are manifold approaches to overcome the jagged boundaries discretization which model the geometries and interfaces through indirect meshing. Some important methods are presented with respect to their compatibility to this work without any claim of completeness. Non-conforming discretization methods gained large interest over the last years, especially in the computational mechanics and computational fluid dynamics but also structural mechanics of voxel-based models from μ CT or MRI image data sets. They prevent the critical process of mesh generation for complex domains,

but they also offer the freedom of choice at the type of structured discretization. One of the first approaches to model complex physical domains or microstructures indirectly was the method of composite finite elements [HS97]. It overlays the regular grid with a virtual aligned grid with piecewise basis function adding a zero weight to the outside domain. For the virtual conforming mesh inspired by marching cubes [LC87] lookup tables were build, which require additional memory. This method was adapted for voxel-based geometries derived by 3D images from MRI or CT and then used for simulation of bone [Pre+11] or foams [Lie+09].

Partition of unity methods [BM97] such as the generalized finite element method for modeling complex geometries [DBO00; MB96; Suk+01] or the extended finite element method [MB02; MDB99; Suk+00; Fuh04; Leg+10] developed for interfaces like cracks and crack growth [SP03; Suk+03; HSP03; Sto+01], damage zones [BT12] and interface failure [HR06] map the geometry through discontinuous enrichment function with additional degrees of freedom. The type of the enrichment function is dependent of the kind of interface and the additional degrees of freedom add to the global stiffness matrix and inflate it's band structure, which makes it numerically more expensive. A good overview of these methods is given in [FB00]. The finite cover method [TK05; TAY03] adapts the manifold method for multiple materials. This basically is a mesh-free method modeling the physical domain inside the regular grid-based mathematical cover domain via interface elements with Lagrange multipliers.

Well known is the method of finite elements with embedded discontinuities for modeling strain or displacement discontinuities e.g. in the fracture process of concrete. Many different approaches have been developed, a comparative study is presented in [Jir00]. Nevertheless these methods have been developed for strain/displacement discontinuities where as in this work the focus is on modeling artificial stress discontinuities arising from jagged boundaries.

Various immersed, embedded or fictitious methods exist which equally embed the complicated domain in a simpler (e.g. Cartesian) fictitious domain. Their main difference is the kind of constraint to enforce the coupling of the rigid body with the embedding domain: Immersed boundary methods impose Dirichlet boundary conditions at the immersed body surface and discretize the immersed surface explicitly. The immersed interface method uses finite differences stencils on Cartesian grids with adaption near the interface [BL92]. This method is very common for solid-fluid interaction.

The method is extended to finite elements as immersed finite element method (IFEM [Zha+04]) which is proposed for the solution of complex fluid and deformable structure interaction problems encountered in many physical models. The solid part is discretized through Lagrange multipliers. Ghost cells and cut-cells are created for the boundary simulation in [HMS11]. The fictitious domain method replaces the complicate domain by a domain-independent mesh as a super-set of the computational domain and is used

for solid-fluid interaction [Glo+99] based on Lagrange multipliers.

The finite cell method applies unusually high order polynomial ansatz function (p-FEM) combined with the ideas of the embedding or fictitious domain methods to structural mechanics [PDR07; Düs+08]. The material boundaries described through level sets are captured through different materials at densely distributed integration points which will be integrated over cells that are independent of the physical domain. A similar approach with low order ansatz function named multiphase elements was proposed by [Lip+97].

As special polynomial ansatz function to model the complicated boundary implicitly B-splines [Häf07; HKK06] and weighted extended B-splines (WEB splines) [HRW01] are used. The support function for B-splines does not fulfill the partition of unity condition, the construction of spline based finite elements leads to a support region larger than one corresponding finite element.

8.2 Embedded boundary method

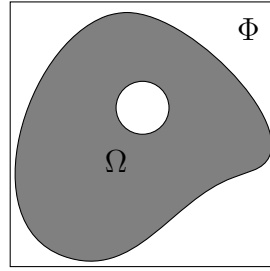


Figure 8.1: The domain Ω is embedded in Φ .

The smoothing of the jagged boundaries is one important point to improve the discontinuities in the stress solution of the voxel-based model. Though retaining the methods presented in this work, the possibilities are rather limited. Creating and storing lookup tables for cut-through elements is quite a memory consuming effort in 3D due the high number of possible ways, yet possible. Considering the indirect methods the finite cell method (FCM) [Düs+08] adapted for lower ansatz-functions seems to be the most suitable approach as the FCM treats all material interface and boundary information in the cell (larger element with embedded boundary) through integration over sub-cells corresponding to the different material information. In the following the embedded boundary/fictitious domain method is reviewed:

A fictitious or embedding domain Φ encloses the physical domain Ω (Fig 8.1), the interface between Ω and the fictitious domain Φ is defined as $\Gamma^\Omega = \delta\Omega \setminus (\delta\Omega \cup \Phi)$. The explaining figures are presented in 2D for their clarity but they apply similarly in 3D. The variational formulation of Eq. (3.40) is now

$$\int_{\Phi} \mathbf{B}^T \alpha \mathbf{C} \mathbf{B} \, d\Phi = \int_{\Phi} N^T \alpha \mathbf{p} \, d\Phi + \int_{\Gamma_f^{\Omega}} N^T \bar{\mathbf{t}} \, d\Gamma_f^{\Omega}. \quad (8.1)$$

in which $\alpha C = C$ is the elasticity matrix of the fictitious domain and α is

$$\alpha(\mathbf{x}) = \begin{cases} 1.0 & \forall \mathbf{x} \in \Omega, \\ 0.0 & \forall \mathbf{x} \in \Phi \setminus \Omega. \end{cases} \quad (8.2)$$

The displacement field is extended following [NT95] as

$$\alpha \mathbf{u} = \begin{cases} \mathbf{u}^1 \in \Omega, \\ \mathbf{u}^2 \in \Phi \setminus \Omega. \end{cases} \quad (8.3)$$

with the condition of continuity at the interface between Ω and $\Phi \setminus \Omega$

$$\mathbf{u}^1 = \mathbf{u}^2 \quad \forall \mathbf{x} \in \Gamma^{\Omega}, \quad (8.4)$$

$$\mathbf{t}^1 = \mathbf{t}^2 \quad \forall \mathbf{x} \in \Gamma^{\Omega} \quad (8.5)$$

and with the additional boundary conditions

$$\bar{\mathbf{u}} = \mathbf{0} \quad \forall \mathbf{x} \in \Gamma_u^{\Phi \setminus \Omega}, \quad (8.6)$$

$$\bar{\mathbf{t}} = \mathbf{0} \quad \forall \mathbf{x} \in \Gamma_f^{\Phi \setminus \Omega}. \quad (8.7)$$

The embedding domain is now discretized independently of the physical domain and elements which embed the material boundary (embedded boundary elements) exist.

8.3 Embedded boundary elements for voxel-based models

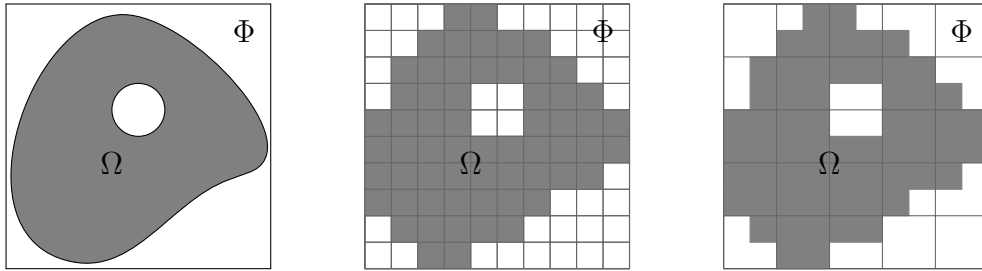


Figure 8.2: The domain Ω embedded in the domain Φ (left), the corresponding pixel image (middle) and the coarse grid with two phases (right)

The discretization applied in this work is solely based on the digital image without boundary detection. This leads to jagged material boundaries and interfaces which causes numerical stress peaks. To overcome this drawback without losing the possibility of matrix-free analysis with few precomputed element stiffness matrices an approach is chosen which does not change the geometry representation but only the integration of the elements with an embedded boundary.

In a first step the model discretization is coarsened by one level as in the multigrid method but for the resulting material properties: the Young's Modulus of each finer element is retained at integration point level and the integration scheme is adapted accounting the discontinuity in the integrand. For this reason the embedded boundary elements are integrated over so-called sub-cells which correspond in this case to the previous finer grid elements which belong to the coarser element. Further adaption of the sub-cell integration scheme to capture the interface is not necessary when the image data directly defines the model geometry. The integration order of the sub-cell is the standard Gauss quadrature scheme for linear hexahedral elements with 8 integration points. The ansatz function in the sub-cell is retained and the stiffness matrix can be integrated exactly with a Gaussian integration of $(p + 1)$ points in each direction with an ansatz function of degree p .

The composed integration of the embedded boundary elements over the eight sub-cells c gives the element stiffness matrix (3.50) of the embedded boundary element m [Düs+08]:

$$\mathbf{K}_m = \sum_{c=1}^8 \sum_{i=1}^{n_{ip}} w^i \mathbf{B}^\top(\boldsymbol{\xi}^i(\boldsymbol{\zeta})) \mathbf{C}^{E_c}(\boldsymbol{\xi}^i(\boldsymbol{\zeta})) \mathbf{B}(\boldsymbol{\xi}^i(\boldsymbol{\zeta})) \det \mathbf{J}_m(\boldsymbol{\xi}^i(\boldsymbol{\zeta})) \times \det \mathbf{J}_m^c(\boldsymbol{\zeta}). \quad (8.8)$$

Due to the change of the variables to the natural coordinates $\boldsymbol{\zeta}$ of each sub-cell, the determinant of the Jacobian matrix \mathbf{J}_m^c has to be multiplied to the element stiffness matrix to establish the mapping between the coordinates of the embedded boundary element (ξ_i) and the sub-cell (ζ_i). The mapping between the regular hexahedral element and sub-cells is a linear function

$$\boldsymbol{\xi} = \begin{bmatrix} \xi_1 + \frac{1}{2}(1 + \zeta_1)h_{\xi_1} \\ \xi_2 + \frac{1}{2}(1 + \zeta_2)h_{\xi_2} \\ \xi_3 + \frac{1}{2}(1 + \zeta_3)h_{\xi_3} \end{bmatrix} \quad (8.9)$$

which results in constant Jacobian matrix.

$$\mathbf{J} = \frac{1}{2} \begin{bmatrix} h_{\xi_1} & 0 & 0 \\ 0 & h_{\xi_2} & 0 \\ 0 & 0 & h_{\xi_3} \end{bmatrix} \quad (8.10)$$

h_{ξ_i} defines the size of the sub-cell in direction i . Through the coarsening of the voxel-

based model the boundary conditions on the surfaces are applied correctly to the embedded boundary model without special considerations.

With the presented embedded boundary adaption the information of the finer grid is retained and the number of dofs is decreased by $1/8$. Furthermore the precomputed element matrix can be reused. However the linear ansatz function leads only to an improvement in the stress discontinuity and not to a smooth solution, in order to achieve this the ansatz function has to be higher [Düs+08].

8.4 Example: 3D plate with circular hole

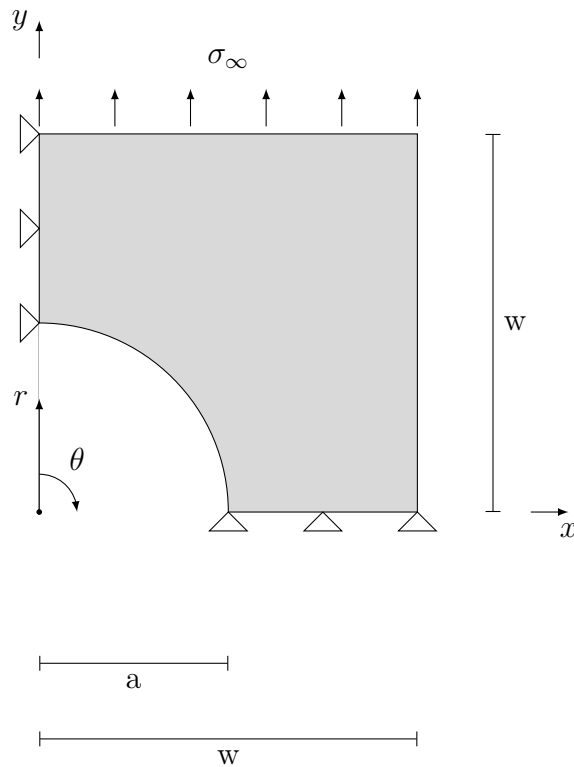


Figure 8.3: Example: Symmetric 3D-plate with circular hole

The 3D-plate with circular hole (Fig. 8.3) with the use of the symmetry axes is considered for the benchmark of the embedded boundary elements. The hole diameter a is $1/4$ of the model width w . The model is loaded with a given uni-axial uniform force $F_z = 1$. The dimension of one voxel is $1mm$ in each direction. The linear elastic material parameters are $E = 100000N/mm^2$ and $\nu = 0,2$. The computation is based on the row-major data structure (Sec. 2.3.2).

For the infinite plate with circular hole an analytical stress solution by Kirsch [Kir98]

exists¹

$$\sigma_\theta = \frac{\sigma_\infty}{2} \left(1 + \left(\frac{a}{r} \right)^2 \right) - \frac{\sigma_\infty}{2} \left(1 + 3 \left(\frac{a}{r} \right)^4 \right) \cos 2\theta \quad (8.11)$$

for a uni-axial loading with a stress concentration factor $K_t = 3$ which is defined as

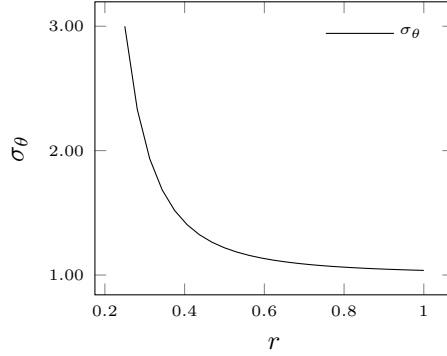


Figure 8.4: Stress reference solution for symmetric plate with hole ($a/w = 0.25$) for $\theta = 90^\circ$

the ratio of maximum stress σ_{max} at a hole to the remote stress σ_∞ . The solution can be adapted for a finite plate considering the nominal stress at the reduced width of the plate σ_{nom} through a empirical formulation [Pil05]:

$$K_t = \frac{\sigma_{max}}{\sigma_{nom}} = 3 - 3.14 \left(\frac{d}{w} \right) + 3.667 \left(\frac{d}{w} \right)^2 - 1.527 \left(\frac{d}{w} \right)^3 \quad (8.12)$$

For the considered *plate* example it yields $K_t = 2.42$ and $\sigma_{max} = 3.23$.

$$\frac{\sigma_{max}}{\sigma_\infty} = K_t \frac{1}{1 - \left(\frac{d}{w} \right)} \quad (8.13)$$

To compare the analytical solution to the example the influence of the third dimension is neglected through the application of plane stress and strain conditions.

8.5 Benchmark

The Fig. 8.5 displays the stress solution in the loading direction for the embedded boundary element model² (left) and the respective original model (right). Fig. 8.6 shows the maximal stress and the stress at $\theta = 90^\circ$ at the hole relative to the analytical solution. As expected the stress solution of the respective embedded boundary model shows reduced maximal stress values compared to the original model. For the highest applied

¹Only the stress solution in loading direction is analyzed.

²The sub-cells of the embedded boundary elements are not illustrated.

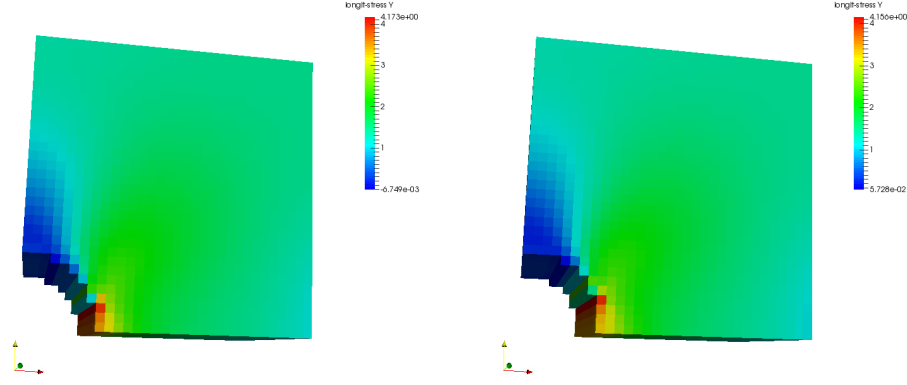


Figure 8.5: Stress σ_y for the embedded boundary element model (left) and the respective single-phase model (right) with 32 voxels per dimension

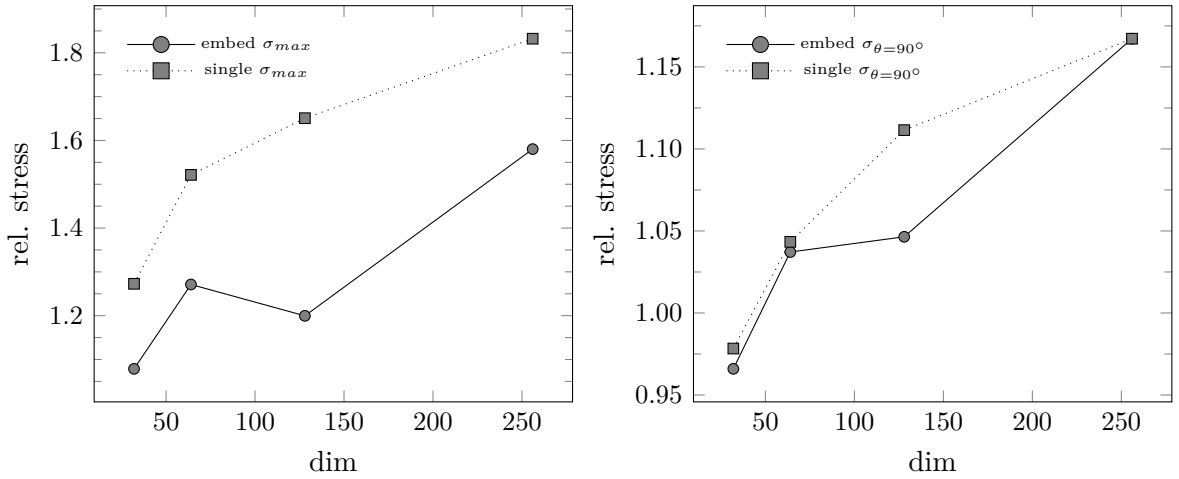


Figure 8.6: Stress σ_{max} (left) and σ for $\theta = 90^\circ$ (right) at the hole for the embedded boundary element model and the original model for the *plate* example relative to the maximal stress $\sigma_{ref} = 3,23$ of the analytical solution

resolution of both models the stress value at the hole for $\theta = 90^\circ$ is overestimated by 15% which is considered acceptable.

In Fig. 8.7 the computation time and the memory demand of the embedded boundary elements and the original finite element model are presented relative to the analysis of the largest original problem without coarsening. The MP models have slightly more dofs than the original model of this discretization. For the same number of degree of freedoms the computational effort is effectively equal as the same integration order or ansatz function is used. The memory demand is higher considering the same resolution caused by the additional material data at the integration points of the embedded element. Though compared to the original model which was the basis of the embedded boundary model the memory demand is about 40% lower. Fig 8.8 shows the same results in function of

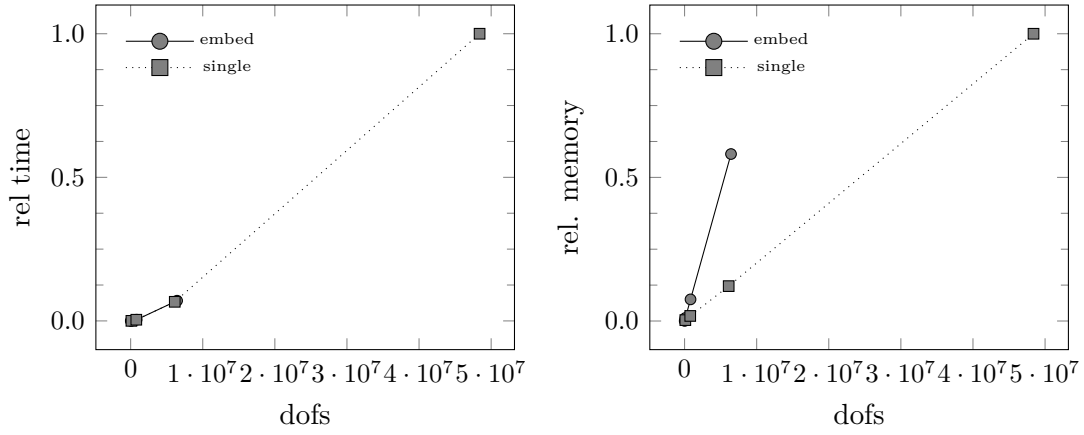


Figure 8.7: Computation time and memory demand vs. dofs of the embedded boundary element and the original model for the *plate* example

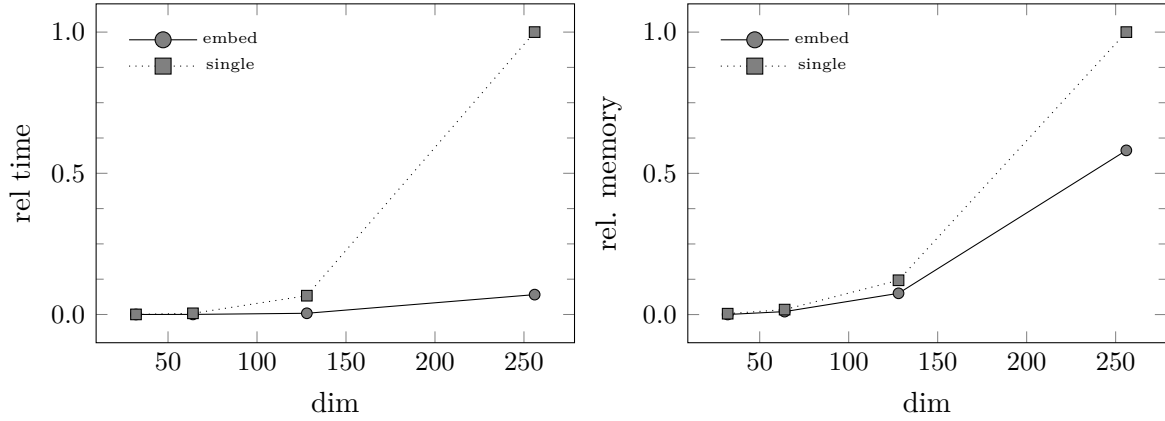


Figure 8.8: Computation time and memory demand vs. dimension of the embedded boundary element and the original model for the *plate* example

the original dimension of the model. The coarsening of the model speeds up the solution process considerably (about 97%) without loss of information.

Chapter 9

Summary, conclusion and outlook

Modern image detection techniques such as MRI, SEM and μ CT provide high resolution information of materials in a non-destructive way. This poses an exceptional basis for micro-mechanical analysis. Nevertheless high resolution image data leads to large numerical models. In this work effective solution algorithms have been adapted and implemented which make such analysis feasible on modern desktop computers while keeping the fine resolution material information. The main restriction is thereby not posed by the central processing unit (CPU) but by the memory requirements. Consequently the reduction of the memory demand is one main aspect in this work.

There are multiple ways to generate a numerical model from image data, applying boundary detection algorithms as basis for aligned meshes or as level set function for indirect meshes such as extended finite element method (XFEM) and finite cell method (FCM) or omitting the boundary detection and creating the numerical model directly from the image data as voxel-based model. The last approach was considered in this work. Voxel-based models use the intrinsic Cartesian structure of the image data. This regular structure allows the application of specific efficient numerical solution methods. The most important is the adaption of the equation system to matrix-free computation which omits the construction, computation and saving of the global stiffness matrix. Therewith analysis are feasible on desktop workstation which would otherwise exceed the hardware's capabilities (speaking of usual desktop PCs) even considering other approaches for the corresponding image data such as aligned meshes. Though in this work a matrix-free solution algorithm was implemented together with efficient iterative solution methods.

For huge linear equation systems iterative methods are the first choice. The conjugate gradient method which is particularly suited for solving equation systems with a positive definite system matrix is applied. The iterative solver was adapted for matrix-free solution methods in two different ways. Solving the equation system with an element-by-element (EBE) or node-edge-based (NE) approach is proposed, implemented and compared using the JCG iterative solver. For the hexahedral mesh (voxel-based), the EBE was found superior to the NE method, especially regarding the memory demand.

The convergence rate of the CG method is dependable of a good preconditioning of the system. Various approaches are not suitable for matrix-free procedures as they require an assembled stiffness matrix. Though, the multigrid algorithm is especially suited for regular meshes such as the voxel-based discretization and can be solved at each level through the same matrix-free means than the basic voxel-based mesh. So the multigrid method is implemented as highly efficient preconditioner of the conjugate gradient method. With these methods a good combination is found to perform numerical analysis of the Cartesian grid models (voxel-based model).

Further decrease of the memory demand without losing image information was the aim of the applied octree algorithm. For this reason a special data structure which sorts the finite elements and nodes through the Morton key was implemented. The model size and therewith the memory demand can be reduced considerably with the coarsening of the homogeneous regions though it is dependable of the heterogeneity of the underlying image data (material). Unfortunately the required neighbor search slows the computation down. Applying an adapted search algorithm [FA11b] further enhanced for an octree discretization led not to a sufficiently speed-up, supposedly through the additional clauses required to prevent *undefined behavior*, though the result of a small analysis was correct.

At last embedded boundary elements were presented which were adapted to especially complement the matrix-free algorithms with the precomputed element stiffness matrix and which do not require boundary detection. Therewith the numerical induced stress discontinuities due to the jagged boundaries of the voxel-based discretization could be reduced.

The introduced methods allow the computation of large image-based models on common desktop PCs. Therewith one major aim of this work could be reached. The matrix-free procedure as the key-feature proved to be very efficient to analyze microstructural models with low memory footprint and could be combined with likewise effective iterative solution methods as the multigrid preconditioned conjugate gradient method.

The applied coarsening with octree data structure could reduce the memory demand but with an increase in computation time due to the neighbor calculation and even more so from the calculation of the constraint hanging nodes. Consequently the homogenous discretization is favored over the reduction of memory gained by octree data structure. Further investigation on the applied lookup table could be interesting. The octree can also be the basis of domain decomposition for MPI parallelization.

The proposed embedded boundary elements could reduce the numerical induced stress peaks at sharp edges caused by the jagged material boundaries and furthermore the problem size is also reduced to about $1/8$ due to the transfer to the coarser grid. Further studies with an adaption of the embedded boundary elements to at least quadratic ansatz functions to achieve more smooth stress behavior could be promising, though the degrees

of freedom will increase and therewith the reduction of the memory demand caused by the initial level change to one coarser level will be lost. Another strategy which could be worth an investigation is to improve the stress field is by subtracting the known value of the stress singularity for a 90° corner.

The presented methods are aimed for modern desktop computer or moderate clusters, which provide a small number of CPUs and shared memory. The chosen methods are well suited for parallelization, so the parallelization with openMP or/and MPI would further speed up the computation [GD01], additional the matrix-free computation is especially suited for parallelization on graphical processing units (GPU) [DGW11; AFK14]. So the parallelization of the proposed methods to speed-up the solution process is an important step for the future.

Due to the small memory footprint of the model and the precomputed reused element stiffness matrix the presented method is well suited for sequential linear analysis to simulate nonlinear material behavior [Sch13].

Appendix A

Encode and decode morton numbers

The algorithms to generate Morton codes are the following:

```
//Encode Morton key in 3D
uint32 EncodeMorton3(uint32 x, uint32 y, uint32 z)
{
    return (Part1By2(z) << 2) + (Part1By2(y) << 1) + Part1By2(x);
}

// "Insert" two 0 bits after each of the 10 low bits of x
uint32 Part1By2(uint32 x)
{
    x &= 0x000003ff; // x = ---- ---- ---- ---- ---- --98 7654 3210
    x = (x ^ (x << 16)) & 0xff0000ff; // x = ---- --98 ---- ---- ---- ---- 7654 3210
    x = (x ^ (x << 8)) & 0x0300f00f; // x = ---- --98 ---- ---- 7654 ---- ---- 3210
    x = (x ^ (x << 4)) & 0x030c30c3; // x = ---- --98 ---- 76-- --54 ---- 32-- --10
    x = (x ^ (x << 2)) & 0x09249249; // x = ---- 9--8 --7- -6-- 5--4 --3- -2-- 1--0
    return x;
}

//Inverse of Part1By2 - "delete" all bits not at positions divisible by 3
uint32 Compact1By2(uint32 x)
{
    x &= 0x09249249; // x = ---- 9--8 --7- -6-- 5--4 --3- -2-- 1--0
    x = (x ^ (x >> 2)) & 0x030c30c3; // x = ---- --98 ---- 76-- --54 ---- 32-- --10
    x = (x ^ (x >> 4)) & 0x0300f00f; // x = ---- --98 ---- ---- 7654 ---- ---- 3210
    x = (x ^ (x >> 8)) & 0xff0000ff; // x = ---- --98 ---- ---- ---- ---- 7654 3210
    x = (x ^ (x >> 16)) & 0x000003ff; // x = ---- ---- ---- ---- ---- --98 7654 3210
    return x;
}
```

Therewith coordinates can be derived from the morton code:

```
//Decode Morton key for x-coordinate
uint32 DecodeMorton3X(uint32 code)
{

```

```
    return Compact1By2(code >> 0);  
}  
  
//Decode Morton key for y-coordinate  
uint32 DecodeMorton3Y(uint32 code)  
{  
    return Compact1By2(code >> 1);  
}  
  
//Decode Morton key for z-coordinate  
uint32 DecodeMorton3Z(uint32 code)  
{  
    return Compact1By2(code >> 2);  
}
```

This code is mainly from the webpage [Gie09].

Appendix B

Bitwise calculation

For the handling of integer with bitwise information bitwise calculation is necessary.

XOR
(exclusive or)

```
0 0 1 ^
1 1 1
-----
1 1 0
```

AND

```
0 0 1 &
1 1 1
-----
0 0 1
```

OR
(inclusive or)

```
0 0 1 |
1 1 1
-----
1 1 1
```

shift left

```
shift 1 << 2:
      1:  0 0 0 1
-----
<<2:  4:  0 1 0 1
```

shift right

```
shift 5 >> 2:
      5:  0 1 0 1
-----
>>2:  1:  0 0 0 1
```

Bibliography

- [ABL16] Alis, C., Boehm, J., and Liu, K. “Parallel processing of big point clouds using Z-Order-based partitioning”. In: vol. 41. 2016, pp. 71–77.
- [AEM06] Akhtar, R., Eichhorn, J., and Mummery, P. “Microstructure-based Finite Element Modelling and Characterisation of Bovine Trabecular Bone”. In: *Journal of Bionic Engineering* 3 (2006), pp. 003–009.
- [Aes08] Aesculap. 2008. URL: <https://www.bbraun.de/de/products/b0/sternumfixation.html>.
- [AFK14] Arbenz, P., Flaig, C., and Kellenberger, D. “Bone structure analysis on multiple GPGPUs”. In: *Journal of Parallel and Distributed Computing* 74.10 (2014), pp. 2941–2950.
- [AGL05] Ahrens, J., Geveci, B., and Law, C. *ParaView: An End-User Tool for Large Data Visualization, Visualization Handbook*. Elsevier, 2005.
- [Arb+07] Arbenz, P., Lenthe, G. H. V., Mennel, U., Müller, R., and Sala, M. “A scalable multi-level preconditioner for matrix-free μ -finite element analysis of human bone structures”. In: *International Journal for Numerical Methods in Engineering* 73.7 (2007), pp. 927–947.
- [ARS06] Augarde, C., Ramage, A., and Staudacher, J. “An element-based displacement preconditioner for linear elasticity problems”. In: *Computers & Structures* 84.31–32 (2006), pp. 2306–2315.
- [Aug+16] Augustin, C., Neic, A., Liebmann, M., Prassl, A., Niederer, S., Haase, G., and Plank, G. “Anatomically accurate high resolution modeling of human whole heart electromechanics: A strongly scalable algebraic multigrid solver method for non-linear deformation”. In: *Journal of Computational Physics* 305 (2016), pp. 622–646.
- [AW06] Adams, M. D. and Wise, D. S. “Fast additions on masked integers”. In: *SIGPLAN Notices* 41.5 (2006), pp. 39–45.
- [Bac66] Bachwalow, N. “On the convergence of a relaxation method with natural constraints on the elliptic operator.” In: *ž. vyčísli. Mat. mat. Fiz.* 6.5 (1966), pp. 861–883.
- [Bad12] Bader, M. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer Publishing Company, Incorporated, 2012.

- [Bat96] Bathe, K. *Finite Element Procedures*. Prentice-Hall, 1996.
- [BD85] Bank, R. E. and Douglas, C. C. “Sharp Estimates for Multigrid Rates of Convergence with General Smoothing and Acceleration”. In: *SIAM Journal on Numerical Analysis* 22.4 (1985), pp. 617–633.
- [Bel99] Belgacem, F. B. “The Mortar finite element method with Lagrange multipliers”. In: *Numerische Mathematik* 84.2 (1999), pp. 173–197.
- [Ben02] Benzi, M. “Preconditioning Techniques for Large Linear Systems: A Survey”. In: *Journal of Computational Physics* 182 (2002), pp. 418–477.
- [BG13] Becker, W. and Gross, D. *Mechanik elastischer Körper und Strukturen*. Springer-Verlag, 2013.
- [BH16] Burstedde, C. and Holke, J. “A tetrahedral space-filling curve for nonconforming adaptive meshes”. In: *SIAM Journal on Scientific Computing* 38.5 (2016), pp. C471–C503.
- [BL92] Beyer, R. P. and LeVeque, R. J. “Analysis of a one-dimensional model for the immersed boundary method”. In: *SIAM J. Numer. Anal.* 29(2) (1992), pp. 332–364.
- [BM97] Babuška, I. and Melenk, J. “The Partition of Unity Method”. In: *International Journal for Numerical Methods in Engineering* 40 (1997), pp. 727–758.
- [Bol+03] Bolz, J., Farmer, I., Grinspun, E., and Schröder, P. “Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid”. In: *ACM SIGGRAPH 2003 Papers*. San Diego, California, 2003.
- [Bra73] Brandt, A. “Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems”. In: *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*. Ed. by Cabannes, H. and Temam, R. Vol. 18. Lecture Notes in Physics. Springer Berlin Heidelberg, 1973, pp. 82–89.
- [Bra77] Brandt, A. “Multi-level adaptive solutions to boundary-value problems”. In: *Mathematics of Computation* 31 (138) (1977), pp. 338–390.
- [Bra86] Braess, D. “On the combination of the multigrid method and conjugate gradients”. In: *Multigrid Methods II*. Ed. by Hackbusch, W. and Trottenberg, U. Vol. 1228. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1986, pp. 52–64.
- [Bra93] Bramble, J. H. *Multigrid Methods*. Essex: Longman, 1993.
- [Bri87] Briggs, W. L. *A Multigrid Tutorial*. 2nd. SIM, 1987.
- [Bro10] Brown, J. “Efficient nonlinear solvers for nodal high-order finite elements in 3D”. In: *Journal of Scientific Computing* 45.1-3 (2010), pp. 48–63.
- [Bru+06] Bruhin, R., Wippermann, J., Franke, U., Breuer, M., Aboud, A., Könke, C., and Wahlers, T. “Stability after median sternotomy - First results applying a computer based closure technique”. In: *Thorac Cardiovasc Surg* 54.Suppl. (2006), S83–S115.

- [BT12] Benvenuti, E. and Tralli, A. “Simulation of finite-width process zone in concrete-like materials by means of a regularized extended finite element model”. In: *Computational Mechanics* 50.4 (2012), pp. 479–497.
- [Cas+08] Castro, R., Lewiner, T., Lopes, H., Tavares, G., and Bordignon, A. “Statistical optimization of octree searches”. In: *Computer Graphics Forum* 27, 6 (2008), pp. 1557–1566.
- [Che+07] Chevalier, Y., Pahr, D., Allmer, H., Charlebois, M., and Zysset, P. “Validation of a voxel-based FE method for prediction of the uniaxial apparent modulus of human trabecular bone using macroscopic mechanical tests and nanoindentation”. In: *Journal of Biomechanics* 40.15 (2007), pp. 3333–3340.
- [Cho+09] Choi, M. G., Ju, E., Chang, J., Kim, Y. J., and Lee, J. “Linkless Octree Using Multi-Level Perfect Hashing”. In: *Computer Graphics Forum (Pacific Graphics 2009)* 28 (8) (2009), pp. 1773–1780.
- [Cou+01] Coutinho, A., Martins, M., Alves, J., Landau, L., and Moraes, A. “Edge-based finite element techniques for non-linear solid mechanics problems”. In: *International Journal for Numerical Methods in Engineering* 50.9 (2001), pp. 2053–2068.
- [Cou+87] Coutinho, A., Alves, J., Landau, L., Lima, E., and Ebecken, N. “On the application of an element-by-element lanczos solver to large offshore structural engineering problems”. In: *Computers & Structures* 27.1 (1987), pp. 27–37.
- [DBO00] Duarte, C. A., Babuska, I., and Oden, J. T. “Generalized finite element methods for three-dimensional structural mechanics problems”. In: *Computers & Structures* 77.2 (2000), pp. 215–232.
- [DGW11] Dick, C., Georgii, J., and Westermann, R. “A real-time multigrid finite hexahedra method for elasticity simulation using CUDA”. In: *Simulation Modelling Practice and Theory* 19.2 (2011), pp. 801–816.
- [Düs+08] Düster, A., Parvizian, J., Yang, Z., and Rank, E. “The finite cell method for three-dimensional problems of solid mechanics”. In: *Computer Methods in Applied Mechanics and Engineering* 197.45-48 (2008), pp. 3768–3782.
- [EK08] Eckardt, S. and Könke, C. “Adaptive damage simulation of concrete using heterogeneous multiscale models”. In: *Journal of Algorithms & Computational Technology* 2.2 (2008), pp. 275–297.
- [EMC06] Elias, R., Martins, M., and Coutinho, A. “Parallel edge-based solution of viscoplastic flows with the SUPG/PSPG formulation”. In: *Computational Mechanics* 38.4-5 (2006), pp. 365–381.
- [FA11a] Flaig, C. and Arbenz, P. “A highly scalable matrix-free multigrid solver for μ FE analysis based on a pointer-less octree”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7116 LNCS (2011), pp. 498–506.

- [FA11b] Flaig, C. and Arbenz, P. “A scalable memory efficient multigrid solver for micro-finite element analyses based on CT images”. In: *Parallel Computing* 37.12 (2011), pp. 846–854.
- [FA12] Flaig, C. and Arbenz, P. “A Highly Scalable Matrix-Free Multigrid Solver for μ FE Analysis Based on a Pointer-Less Octree”. In: *Large-Scale Scientific Computing: 8th International Conference, LSSC 2011, Sozopol, Bulgaria, June 6-10, 2011, Revised Selected Papers*. Ed. by Lirkov, I., Margenov, S., and Waśniewski, J. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 498–506.
- [FB00] Fries, T.-P. and Belytschko, T. “The extended / generalized finite element method: An overview of the method and its applications”. In: *Int. J. Numer. Meth. Engng* 00, 1-6 (2000).
- [FD16] Fambri, F. and Dumbser, M. “Spectral semi-implicit and space-time discontinuous Galerkin methods for the incompressible Navier-Stokes equations on staggered Cartesian grids”. In: *Applied Numerical Mathematics* 110 (2016), pp. 41–74.
- [Fed62] Fedorenko, R. “A relaxation method for solving elliptic difference equations”. In: *USSR Computational Mathematics and Mathematical Physics* 1.4 (1962), pp. 1092–1096.
- [Fuh04] Fuhlrott, A. *Implementierung der X-FEM in SLang und Verifizierung der Anwendbarkeit für Rissberechnungen*. Diplom thesis, Weimar, Germany, 2004.
- [Gao+17] Gao, S., Habashi, W., Isola, D., Baruzzi, G., and Fossati, M. “A Jacobian-free Edge-based Galerkin Formulation for Compressible Flows”. In: *Computers and Fluids* 143 (2017), pp. 141–156.
- [Gar82a] Gargantini, I. “An effective way to represent quadtrees”. In: *Communications of the ACM* 25 (1982), pp. 905–910.
- [Gar82b] Gargantini, I. “Linear octrees for fast processing of three-dimensional objects”. In: *Computer Graphics and Image Processing* 20.4 (1982), pp. 365–374.
- [GD01] Gullerud, A. and Dodds, R. J. “MPI-based implementation of a PCG solver using an EBE architecture and preconditioner for implicit, 3-D finite element analysis”. In: *Computers and Structures* 79 (2001), pp. 553–575.
- [Gie09] Giesen, F. *Decoding Morton Codes*. Dec. 2009. URL: <http://fgiesen.wordpress.com/2009/12/13/decoding-morton-codes/>.
- [GKA07] Giannopoulos, G., Karagiannis, D., and Anifantis, N. “Micromechanical modeling of mechanical behavior of Ti-6Al-4V/TiB composites using FEM analysis”. In: *Computational Materials Science* 39.2 (2007), pp. 437–445.
- [Glo+99] Glowinski, R., Pan, T.-W., Hesla, T., and Joseph, D. “A distributed Lagrange multiplier/fictitious domain method for particulate flows”. In: *International Journal of Multiphase Flow* 25.5 (1999), pp. 755–794.

- [Grö+09] Gröning, F., Liu, J., Fagan, M., and O’Higgins, P. “Validating a voxel-based finite element model of a human mandible using digital speckle pattern interferometry”. In: *Journal of Biomechanics* 42 Issue.9 (2009), pp. 1224–1229.
- [GWX16] Grasedyck, L., Wang, L., and Xu, J. “A nearly optimal multigrid method for general unstructured grids”. In: *Numerische Mathematik* 134.3 (2016), pp. 637–666.
- [GZ99] Griebel, M. and Zumbusch, G. “Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves”. In: *Parallel Computing* 25.7 (1999), pp. 827–843.
- [Hac85] Hackbusch, W. *Multi-Grid Methods and Applications*. Springer-Verlag Berlin, 1985.
- [Hac91] Hackbusch, W. *Iterative Lösung großer schwachbesetzter Gleichungssysteme*. Teubner, 1991.
- [Häf+06] Häfner, S., Eckardt, S., Luther, T., and Könke, C. “Mesoscale modeling of concrete: Geometry and numerics.” In: *Computers and Structures* 84.7 (2006), pp. 450–461.
- [Häf07] Häfner, S. “Grid-based procedures for the mechanical analysis of heterogeneous solids”. PhD thesis. Bauhaus-Universität Weimar, 2007.
- [Hil91] Hilbert, D. “Über die stetige Abbildung einer Line auf ein Flächenstück”. In: *Mathematische Annalen* 38 (3) (1891), pp. 459–460.
- [HK06] Häfner, S. and Könke, C. “Multigrid preconditioned conjugate gradient method in the mechanical analysis of heterogeneous solids”. In: *Proceedings of the 17th International Conference on the Application of Computer Science and Mathematics in Architecture and Civil Engineering*. Ed. by Gürlebeck, K. and Könke, C. Weimar, Germany, 2006.
- [HK94] Hollister, S. J. and Kikuchi, N. “Homogenization theory and digital imaging: A basis for studying the mechanics and design principles of bone tissue”. In: *Biotechnology and Bioengineering* 43.7 (1994), pp. 586–596.
- [HKK06] Häfner, S., Kessel, M., and Könke, C. “Multiphase b-spline finite elements of variable order in the mechanical analysis of heterogenous solids”. In: *Proceedings of the 17th International Conference on the Application of Computer Science and Mathematics in Architecture and Civil Engineering (IKM 2006)*. Ed. by Gürlebeck, K. and Könke, C. Weimar, Germany, 2006.
- [HL99] Heinsteins, M. and Laursen, T. “An algorithm for the matrix-free solution of quasistatic frictional contact problems”. In: *International Journal for Numerical Methods in Engineering* 44.9 (1999), pp. 1205–1226.
- [HLW83] Hughes, T., Levit, I., and Winget, J. “An element-by-element solution algorithm for problems of structural and solid mechanics”. In: *Computer Methods in Applied Mechanics and Engineering* 36.2 (1983), pp. 241–254.
- [HMS08] Hartmann, D., Meinke, M., and Schröder, W. “An adaptive multilevel multigrid formulation for Cartesian hierarchical grid methods”. In: *Computers and Fluids* 37.9 (2008), pp. 1103–1125.

- [HMS11] Hartmann, D., Meinke, M., and Schröder, W. “A strictly conservative Cartesian cut-cell method for compressible viscous flows on adaptive grids”. In: *Computer Methods in Applied Mechanics and Engineering* 200.9:12 (2011), pp. 1038–1052.
- [HR06] Hettich, T. and Ramm, E. “Interface material failure modeled by the extended finite-element method and level sets”. In: *Computer Methods in Applied Mechanics and Engineering* 195 (2006), pp. 4753–4767.
- [HRW01] Höllig, K., Reif, U., and Wipper, J. “Weighted extended b-spline approximation of Dirichlet problems”. In: *SIAM Journal on Numerical Analysis* 39.2 (2001), pp. 442–462.
- [HS52] Hestenes, M. R. and Stiefel, E. “Methods of Conjugate Gradients for Solving Linear Systems”. In: *Journal of Research of the National Bureau of Standards* 49.6 (1952).
- [HS97] Hackbusch, W. and Sauter, S. “Composite finite elements for the approximation of PDEs on domains with complicated micro-structures”. In: *Numerische Mathematik* 75 (4 1997), pp. 447–472.
- [HSP03] Huang, R., Sukumar, N., and Prévost, J.-H. “Modeling Quasi-Static Crack Growth with the Extended Finite Element Method. Part II: Numerical Applications”. In: *International Journal of Solids and Structures* (2003).
- [HW16] Hussain, N. and Webb, J. “Preconditioners for the nonconforming voxel edge element method”. In: 2016.
- [HZ15] Hofreither, C. and Zulehner, W. “Spectral analysis of geometric multigrid methods for isogeometric analysis”. In: *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8962 (2015), pp. 123–129.
- [Ima+12] Imakura, A., Sakurai, T., Sumiyoshi, K., and Matsufuru, H. “An Auto-Tuning Technique of the Weighted Jacobi-Type Iteration Used for Preconditioners of Krylov Subspace Methods”. In: *IEEE 6th International Symposium on Embedded Multi-core Socs (MCSoc)* (2012), pp. 183–190.
- [ISS08] Ito, Y., Shih, A. M., and Soni, B. K. “Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates”. In: *International Journal for Numerical Methods in Engineering* (2008).
- [Jir00] Jirašek, M. “Comparative study on finite elements with embedded discontinuities”. In: *Computer Methods in Applied Mechanics and Engineering* 188.1 (2000), pp. 307–330.
- [Jog02] Jog, C. S. *Foundations and applications of mechanics : Vol. 1 Continuum mechanics*. Boca Raton, 2002.
- [Joh+] Johnson, H. J., McCormick, M. M., Ibanez, L., and Consortium, I. S. *ITk - Insight Segmentation and Registration Toolkit*. www.itk.org. URL: www.itk.org.

- [Ket82] Kettler, R. “Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods”. In: *Multigrid Methods*. Ed. by Hackbusch, W. and Trottenberg, U. Vol. 960. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1982, pp. 502–534.
- [Key+90] Keyak, J., Meagher, J., Skinner, H., Mote, C., and Jr. “Automated three-dimensional finite element modelling of bone: a new method”. In: *Journal of Biomedical Engineering* 12.5 (1990), pp. 389–397.
- [Kir98] Kirsch, E. “Die Theorie der Elastizität und die Bedürfnisse der Festigkeitslehre,” in: *Zeitschrift des Vereines deutscher Ingenieure* 42 (1898), pp. 797–807.
- [KK12a] Keßler, A. and Könke, C. “Solving linear sparse systems on three dimensional structured grids using matrix-free methods: A comparison”. In: *ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers* (2012), pp. 2394–2406.
- [KK12b] Kronbichler, M. and Kormann, K. “A generic interface for parallel cell-based finite element operator application”. In: *Computers and Fluids* 63 (2012), pp. 135–147.
- [KS03] Kim, H. J. and Swan, C. C. “Voxel-based meshing and unit-cell analysis of textile composites”. In: *International Journal for Numerical Methods in Engineering* 56 (2003), pp. 977–1006.
- [KW16] Kronbichler, M. and Wall, W. “A performance comparison of continuous and discontinuous Galerkin methods with multigrid solvers, including a new multigrid scheme for HDG”. In: *SIAM Journal on Scientific Computing*.2016 (2016).
- [LC87] Lorensen, W. E. and Cline, H. E. “Marching cubes: A high resolution 3D surface construction algorithm”. In: *SIGGRAPH Comput. Graph.* 21.4 (1987), pp. 163–169.
- [Leg+10] Legrain, G., Cartraud, P., Perreard, I., and Moës, N. “An X-FEM and level set computational approach for image-based modelling: Application to homogenization”. In: *International Journal for Numerical Methods in Enineering* (2010).
- [Lew+10] Lewiner, T., Mello, V., Peixoto, A., Pesco, S., and Lopes, H. “Fast generation of pointerless octree duals”. In: *Symposium on Geometry Processing 2010 (Computer Graphics Forum)*. Vol. 29. 5. Lyon, France: Wiley, 2010, pp. 1661–1669.
- [Li17] Li, W. “A matrix-free, implicit finite volume lattice Boltzmann method for steady flows”. In: *Computers and Fluids* 148 (2017), pp. 157–165.
- [Lie+09] Liehr, F., Preusser, T., Rumpf, M., Sauter, S., and Schwen, L. O. “Composite Finite Elements for 3D Image Based Computing”. In: *Computing and Visualization in Science* 12.4 (2009), pp. 171–188.
- [Lin+14] Lintermann, A., Schlimpert, S., Grimm, J., Günther, C., Meinke, M., and Schröder, W. “Massively parallel grid generation on HPC systems”. In: *Computer Methods in Applied Mechanics and Engineering* 277 (2014), pp. 131–153.

- [Lip+97] Lippmann, N., Steinkopff, T., Schmauder, S., and Gumbsch, P. “3D-finite-element-modelling of microstructures with the method of multiphase elements”. In: *Computational Materials Science* 9 (1997), pp. 28–35.
- [Liu+14] Liu, S., Li, X., Wang, W., and Liu, Y. “A mixed-grid finite element method with PML absorbing boundary conditions for seismic wave modelling”. In: *Journal of Geophysics and Engineering* 11.5 (2014).
- [Mao+16] Mao, S. S., Li, D., Luo, Y., Syed, Y. S., and Budoff, M. J. “Application of quantitative computed tomography for assessment of trabecular bone mineral density, microarchitecture and mechanical property”. In: *Clinical Imaging* 40.2 (2016), pp. 330–338.
- [MB02] Moës, N. and Belytschko, T. “Extended finite element method for cohesive crack growth”. In: *Engineering Fracture Mechanics* 69.7 (2002), pp. 813–833.
- [MB96] Melenk, J. and Babušk, I. “The Partition of Unity Finite Element Method: Basic Theory and Applications”. In: *Computer Methods in Applied Mechanics and Engineering* 139 (1996), 289–314.
- [MBL15] May, D. A., Brown, J., and Le Pourhiet, L. “A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous Stokes flow”. In: *Computer Methods in Applied Mechanics and Engineering* 290 (2015), pp. 496–523.
- [MDB99] Moës, N., Dolbow, J., and Belytschko, T. “A Finite Element Method for Crack Growth Without Remeshing”. In: *International Journal for Numerical Methods in Engineering* 46.1 (1999), pp. 131–150.
- [Mei05] Meister, A. *Numerik linearer Gleichungssysteme - Eine Einführung in moderne Verfahren*. 2. Auflage. Wiesbaden: Vieweg Verlag, 2005.
- [Mis05] Mishnaevsky, L. “Automatic voxel-based generation of 3D microstructural FE models and its application to the damage analysis of composites”. In: *Materials Science and Engineering: A* 407.1-2 (2005), pp. 11–23.
- [MMH15] Martínez-Frutos, J., Martínez-Castejón, P., and Herrero-Pérez, D. “Fine-grained GPU implementation of assembly-free iterative solver for finite element problems”. In: *Computers and Structures* 157 (2015), pp. 9–18.
- [Mor66] Morton, G. M. *A computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. Tech. rep. Ottawa, Canada: IBM Ltd., 1966.
- [MS01] Mishnaevsky, L. and Schmauder, S. “Continuum mesomechanical finite element modeling in materials development: A state-of-the-art review”. In: *American Society of Mechanical Engineers* 54.1 (2001), pp. 49–74.
- [MS05] Mehta, D. P. and Sahni, S. *Handbook of Data Structures and Applications*. Chapman Hall, 2005.
- [MWZ06] Mehl, M., Weinzierl, T., and Zenger, C. “A cache-oblivious self-adaptive full multi-grid method”. In: *Numerical Linear Algebra with Applications* 13.2-3 (2006), pp. 275–291.

- [NCG05] Nagaraja, S., Couse, T. L., and Guldberg, R. E. “Trabecular bone microdamage and microstructural stresses under uniaxial compression”. In: *Journal of Biomechanics* 38.4 (2005), pp. 707–716.
- [NT95] Neittaanmäki, P. and Tiba, D. “An Embedding of Domains Approach in Free Boundary Problems and Optimal Design”. In: *SIAM Journal on Control and Optimization* 33.5 (1995), pp. 1587–1602.
- [NW00] Nievergelt, J. and Widmayer, P. “Spatial data structures: Concepts and design choices”. In: Sack, J.-R. and Urrutia, J. *Handbook of Computational Geometry*. B.V. North-Holland, Amsterdam: Elsevier Science, 2000, pp. 725–764.
- [PB03] Perilli, E. and Baruffaldi, F. “Proposal for shared collections of X-ray microCT datasets of bone specimens”. In: *ICCB03*. Zaragoza, Spain, 2003.
- [PDR07] Parvizian, J., Düster, A., and Rank, E. “Finite cell method: hh- and p-extension for embedded domain problems in solid mechanics”. In: *Computational Mechanics* 41.1 (2007), pp. 121–133.
- [Pea90] Peano, G. “Sur une courbe, qui remplit toute une aire plane”. In: *Mathematische Annalen* 36 (1) (1890), pp. 157–160.
- [Pil05] Pilkey, W. D. *Formulas for Stress, Strain, and Structural Matrices*. 2nd. John Wiley & Sons, 2005.
- [Pre+11] Preusser, T., Rumpf, M., Sauter, S., and Schwen, L. O. “3D Composite Finite Elements for Elliptic Boundary Value Problems with Discontinuous Coefficients”. In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2115–2143.
- [PZ09] Pahr, D. H. and Zysset, P. K. “A comparison of enhanced continuum FE with micro FE models of human vertebral bodies”. In: *Journal of Biomechanics* 42.4 (2009), pp. 455–462.
- [RC05] Ribeiro, F. and Coutinho, A. “Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses”. In: *International Journal for Numerical Methods in Engineering* 63.4 (2005), pp. 569–588.
- [Rie+95] Rietbergen, B. van, Weinans, H., Huiskes, R., and Odgaard, A. “A new method to determine trabecular bone elastic properties and loading using micromechanical finite-element models”. In: *Journal of Biomechanics* 28.1 (1995), pp. 69–81.
- [Rie+96] Rietbergen, B. van, Weinans, H., Huiskes, R., and Polman, B. J. W. “Computational strategies for iterative solutions of large fem applications employing voxel data”. In: *International Journal for Numerical Methods in Engineering* 39.16 (1996), pp. 2743–2767.
- [Rue+12] Ruess, M., Varduhn, V., Rank, E., and Yosibash, Z. “A parallel high-order fictitious domain approach for biomechanical applications”. In: 2012, pp. 279–285.
- [RWR93] Rietbergen, B. van, Weinans, H., and Rik, H. “Three dimensional analysis of a realistic trabecular bone structure, using a large-scale FE-model”. In: *BED-Vol. 24, 1993 Bioengineering Conference ASME*. 1993.

- [Saa03] Saad, Y. *Iterative Methods for Sparse Linear Systems, Second Edition*. a: SIAM, 2003.
- [Sam06] Samet, H. *Foundations of Multidimensional and Metric Data Structures*. First Edition. Morgan Kaufmann, 2006.
- [Sam84] Samet, H. “The Quadtree and Related Hierarchical Data Structures”. In: *ACM Comput. Surv.* 16.2 (June 1984), pp. 187–260.
- [Sam89] Samet, H. “Neighbor Finding in Images Represented by Octrees”. In: *Computer Vision, Graphics and Image Processing* 46 (1989), pp. 367–386.
- [Sam94] Samet, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA., 1994.
- [Sap+17] Saputra, A., Talebi, H., Tran, D., Birk, C., and Song, C. “Automatic image-based stress analysis by the scaled boundary finite element method”. In: *International Journal for Numerical Methods in Engineering* 109.5 (2017), pp. 697–738.
- [SB10] Sampath, R. and Biros, G. “A parallel geometric multigrid method for finite elements on octree meshes”. In: *SIAM Journal on Scientific Computing* 32.3 (2010), pp. 1361–1392.
- [Sch13] Schrader, K. “Hybrid 3D simulation methods for the damage analysis of multiphase composites”. PhD thesis. Bauhaus-Universität Weimar, 2013.
- [SH94] Sagan, H. and Holbrook, J. *Space-Filling Curves*. Springer-Verlag New York, 1994.
- [SML] Schroeder, W. J., Martin, K. M., and Lorensen, W. E. *Visualization Toolkit (VTK)*. www.vtk.org. URL: www.vtk.org.
- [SP03] Sukumar, N. and Prévost, J.-H. “Modeling Quasi-Static Crack Growth with the Extended Finite Element Method. Part I: Computer Implementation”. In: *International Journal of Solids and Structures* (2003).
- [Spe04] Spencer, A. J. M. *Continuum mechanics*. 2nd. Mineola, N.Y : Dover Publ, 2004.
- [SSB15] Saha, P., Strand, R., and Borgefors, G. “Digital Topology and Geometry in Medical Imaging: A Survey”. In: *IEEE Transactions on Medical Imaging* 34.9 (2015), pp. 1940–1964.
- [Sti16] Stiller, J. “Nonuniformly Weighted Schwarz Smoothers for Spectral Element Multigrid”. In: *Journal of Scientific Computing* (2016), pp. 1–16.
- [Sto+01] Stolarska, M., Chopp, D., Moës, N., and Belytschko, T. “Modelling crack growth by level sets in the extended finite element method”. In: *International Journal for Numerical Methods in Engineering* 51.8 (2001), pp. 943–960.
- [Str16] Strang, G. *Introduction to Linear Algebra, Fifth Edition*. Wellesley-Cambridge Press and SIAM, 2016.
- [Suk+00] Sukumar, N., Moës, N., Moran, B., and Belytschko, T. “Extended Finite Element Method for Three-Dimensional Crack Modelling”. In: *International Journal for Numerical Methods in Engineering* 48.11 (2000), pp. 1549–1570.

- [Suk+01] Sukumar, N., Chopp, L., Moës, N., and Belytschko, T. “Modeling Holes and Inclusions by Level Sets in the Extended Finite Element Method”. In: *Computer Methods in Applied Mechanics and Engineering* 190.46-47 (2001), pp. 6183–6200.
- [Suk+03] Sukumar, N., Srolovitz, D., Baker, T., and Prévost, J.-H. “Brittle Fracture in Polycrystalline Microstructures with the Extended Finite Element Method”. In: *International Journal for Numerical Methods in Engineering* 56.14 (2003), pp. 2015–2037.
- [Tat93] Tatebe, O. *The Multigrid Preconditioned Conjugate Gradient Method*. 1993.
- [TAY03] Terada, K., Asai, M., and Yamagishi, M. “Finite cover method for linear and non-linear analyses of heterogeneous solids”. In: *International Journal for Numerical Methods in Engineering* 58.9 (2003), pp. 1321–1346.
- [TK05] Terada, K. and Kurumatani, M. “An integrated procedure for three-dimensional structural analysis with the finite cover method”. In: *International Journal for Numerical Methods in Engineering* 63.15 (2005), pp. 2102–2123.
- [TO94] Tatebe, O. and Oyanagi, Y. “Efficient implementation of the multigrid preconditioned conjugate gradient method on distributed memory machines”. In: 1994, pp. 194–203.
- [TPG98] Treece, G. M., Prager, R. W., and Gee, A. H. “Regularised Marching Tetrahedra: Improved Iso-Surface Extraction”. In: *Computers and Graphics* 23 (1998), pp. 583–598.
- [TW00] Teng, S.-H. and Wong, C. W. “Unstructured mesh generation: Theory, practice and applications.” In: *Internat. J. Comput. Geom. Appl.* 10(3) (2000), pp. 227–266.
- [Wan07] Wang, X. S. “An iterative matrix-free method in implicit immersed boundary/continuum methods”. In: *Computers and Structures* 85.11-14 (2007), pp. 739–748.
- [Wes92] Wesseling, P. *An Introduction to Multigrid Methods*. Chichester: Wiley, 1992.
- [Yan+12] Yang, Z., Ruess, M., Kollmannsberger, S., Düster, A., and Rank, E. “An efficient integration technique for the voxel-based finite cell method”. In: *International Journal for Numerical Methods in Engineering* 91.5 (2012), pp. 457–471.
- [Zan+15] Zander, N., Bog, T., Kollmannsberger, S., Schillinger, D., and Rank, E. “Multi-Level hp-Adaptivity: High-Order Mesh Adaptivity without the Difficulties of Constraining Hanging Nodes”. In: *Computational Mechanics* submitted (2015).
- [Zha+04] Zhang, L., Gerstenberger, A., Wang, X., and Liu, W. K. “Immersed finite element method”. In: *Computer Methods in Applied Mechanics and Engineering* 193.21-22 (2004), pp. 2051–2067.
- [Zoh15] Zohdi, T. “Modeling and Simulation of Coupled Cell Proliferation and Regulation in Heterogeneous Tissue”. In: *Annals of Biomedical Engineering* 43.7 (2015), pp. 1666–1679.

- [Zoh17] Zohdi, T. “Modeling and Simulation of Laser Processing of Particulate-Functionalized Materials”. In: *Archives of Computational Methods in Engineering* 24.1 (2017), pp. 89–113.
- [ZTZ05] Zienkiewicz, O. C., Taylor, R. L., and Zhu, J. Z. *The Finite Element Method: Its Basis and Fundamentals, Sixth Edition*. 6th ed. Butterworth-Heinemann, May 2005.